Randomized Derivative Free Optimization via CMA-ES and Sparse Techniques - Applications to Radars

Konstantinos Varelas

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAX012

Thèse de doctorat

Randomized Derivative Free Optimization via CMA-ES and Sparse Techniques - Applications to Radars

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'Ecole Polytechnique

École doctorale n°574 Ecole Doctorale de Mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques appliquées

Thèse présentée et soutenue à Paris (visio conférence), le 12/02/2021, par

M. KONSTANTINOS VARELAS

Composition du Jury :

Alexandre D'Aspremont Professor, Inria Paris, École Normale Supérieure (Département d'informatique)	Président
Peter Bosman Professor, Delft University of Technology (Department of software technology)	Rapporteur
Silvère Bonnabel Professor, University of New-Caledonia and Mines ParisTech	Rapporteur
Erwan Le Pennec Professor, IP Paris (CMAP)	Examinateur
Cyril Furtlehner CR, Inria Saclay (Laboratoire de recherche en informatique)	Examinateur
Anne Auger DR, Inria Saclay, IP Paris (CMAP)	Directrice de thèse
Dimo Brockhoff CR, Inria Saclay, IP Paris (CMAP)	Co-directeur de thèse
Frédéric Barbaresco Thales	Invité
Yann Semet Thales	Invité


CMAP


Fondation mathématique
FMJH
Jacques Hadamard

Abstract

In this thesis, we investigate aspects of adaptive randomized methods for black-box continuous optimization. The algorithms that we study are based on the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm and focus on large scale optimization problems.

We start with a description of CMA-ES and its relation to the Information Geometric Optimization (IGO) framework, succeeded by a comparative study of large scale variants of CMA-ES. We furthermore propose novel methods which integrate tools of high dimensional estimation within CMA-ES, to obtain more efficient algorithms for large scale partially separable problems.

Additionally, we describe the methodology for algorithm performance evaluation adopted by the Comparing Continuous Optimizers (COCO) platform, and finalize the `bbob-largescale` test suite, a novel benchmarking suite with problems of increased dimensions and with a low computational cost.

Finally, we present the formulation, methodology and obtained results for two applications related to Radar problems, the Phase Code optimization problem and the Phased-Array Pattern design problem.

Resumé

Dans cette thèse, nous étudions différents aspects des méthodes aléatoires adaptatives pour l'optimisation continue de type boîte noire. Les algorithmes que nous étudions sont basés sur l'adaptation de la matrice de variance-covariance d'une stratégie évolutionnaire (CMA-ES). Cet algorithme est considéré comme l'état de l'art pour l'optimisation dans un domaine continu lorsque le gradient n'est pas accessible ou coûteux à évaluer. Il effectue une recherche stochastique en échantillonnant les points candidats selon une distribution gaussienne multivariée qui est mise à jour à chaque itération.

Notre principale motivation est la résolution d'applications au traitement du signal radar, qui sont généralement formulées comme des problèmes d'optimisation non-convexe dans un espace de recherche continu et de grande dimension, c'est pourquoi nous nous intéressons particulièrement aux algorithmes d'optimisation qui sont efficaces dans ce contexte.

Après avoir défini le cadre de l'optimisation continue boîte noire et les différents types d'algorithmes opérant dans ce contexte dans le premier chapitre, on donnera une description de l'algorithme CMA-ES et ses connexions avec l'optimisation géométrique de l'information (IGO) dans le second chapitre. Cette théorie fournit un cadre cohérent et générique pour adapter une distribution de probabilité dans l'espace de recherche, l'utilisation de méthodes stochastique étant populaire pour les problèmes d'optimisation boîte noire. La description de CMA-ES et IGO est suivie d'une étude comparative des variantes de CMA-ES pour la grande dimension. On s'attardera à comparer les variantes de CMA-ES prometteuses qui ont été proposées ces dernières années, en discutant de leurs avantages et de leurs limites à partir de l'observation des résultats de l'analyse comparative expérimentale.

Dans le troisième chapitre, nous proposons de nouvelles méthodes intégrant des outils d'estimation de la matrice de variance-covariance en grande dimension, tels que la régularisation graphical lasso ou la technique de hard-thresholding afin d'accélérer l'optimisation avec CMA-ES en grande dimension pour des problèmes d'une certaine classe. En particulier, trois méthodes différentes sont proposées et ont pour objectif commun d'accélérer la vitesse d'adaptation de la matrice de covariance lors de l'optimisation de fonctions objectifs appartenant à la classe de fonctions dite partiellement séparable. Les méthodes tentent essentiellement d'exploiter les propriétés de parcimonie de telles fonctions, ce qui permet une adaptation plus rapide de la distribution de recherche.

Le quatrième chapitre est divisé en deux parties: la première partie décrit la méthodologie adoptée par la plate-

forme Comparing Continuous Optimizers (COCO) pour évaluer les performances d'un algorithme d'optimisation, présente les spécificités liés à l'évaluation des performances en grande dimension et introduit un ensemble de problèmes tests implémentés avec COCO dans la suite `bbob-largescale`. Cette suite est construite dans un esprit similaire à la suite `bbob` et examine des problèmes en plus grande dimension tout en maintenant avec un faible coût de calcul. La deuxième partie du chapitre est constituée de deux études reposant sur des résultats expérimentaux pour plusieurs algorithmes d'optimisation en considérant un large champ de dimension (utilisant respectivement les suites `bbob` et `bbob-largescale`).

Le cinquième chapitre est dédié à la résolution de deux applications radar: le problème de la recherche de codes de phase pour le filtrage adapté et celui de la synthèse des faisceaux dans une antenne réseau à commande de phase (Phased-Array antenna). Le premier est un problème de conception de forme d'onde pour les radars MIMO (Multiple Input Multiple Output) minimisant certaines propriétés de corrélation. Le second est un problème de synthèse des faisceaux de chaque antenne élémentaire contenue dans une antenne réseau à commande de phase pour obtenir un faisceau de forme non standard. Dans les deux cas, la difficulté vient de la grande dimension et de la multimodalité de la fonction objectif. Nous proposons des méthodes efficaces basées sur CMA-ES afin d'obtenir des solutions de bonne qualité et nous illustrons les résultats obtenus dans différents cas test.

Enfin, dans le dernier chapitre, nous faisons le bilan des approches considérées au cours de la thèse, et nous proposons également des directions de recherche pour les améliorer davantage.

Acknowledgements

At the point of finishing this dissertation, I would like to sincerely thank the people who have supported me during the period of my studies.

First and foremost, I would like to express my gratitude to my advisors Anne Auger, Dimo Brockhoff, Rami Kassab and Yann Semet for their help and guidance, which was essential for the fulfillment of this dissertation. Under their supervision, I had the opportunity to heartily enjoy their academic care simultaneously with academic freedom. I would also like to thank the RandOpt team members: Nikolaus Hansen, Cheikh Touré, Paul Dufossé, Eugénie Marescaux, Marie-Ange Dahito, Alann Cheral and Tea Tušar as well as my colleagues Frédéric Barbaresco, Yann Briheche and Johann Dreo who gave me motivation with their sincere interest, energy and curiosity.

I am especially grateful to the members of my Ph.D. thesis committee, Prof. Peter Bosman (Delft University of Technology), Prof. Silvère Bonnabel (University of New-Caledonia Noumea), Prof. Erwan Le Pennec (IP Paris), Prof. Alexandre D'Aspremont (École Normale Supèrieure) and Dr. Cyril Furtlehner (Inria Saclay) for accepting to review and evaluate this work.

My academic journey started at the National Technical University of Athens as an undergraduate student, followed by postgraduate studies at the Pierre et Marie Curie and Paris Saclay Universities. I am thankful for the excellent academic environment that I met, and I would like to especially thank my professors Ioannis Spiliotis, Nikolaos Maratos, Dimitris Apatsidis and Filippo Santambrogio. Each one of them from diverse scientific domains gave me a very strong motivation in order to pursue my studies.

Furthermore, I sincerely thank the people closest to me who supported me during the period of the thesis: Marie, Cleopatra, Pavlos, Kostas, Giorgos, Emma and my old friends Nikos, Angelos, Giannis, Dimitris and Alexis. Last but not least, I deeply thank my family members and close relatives Christos, Katerina, Lamprini, Panagiota, Sofia, Panagiotis, Konstantinos, Lamprini, Rosa, Giannis, Mary and Panagiotis. Thank you for everything that you have done and continue doing for me.

This PhD thesis has been funded by the French MoD DGA/MRIS and Thales Land & Air Systems.

To my family

Contents

Notations and Acronyms

aRT: average runtime

a.s.: almost surely

CMA-ES: Covariance Matrix Adaptation Evolution Strategy

COCO: Comparing Continuous Optimizers

CSA: Cumulative Step-size Adaptation

DFT: Discrete Fourier Transform

dB: decibel

ECDF: Empirical Cumulative Distribution Function

ERT: Expected runtime

FFT: Fast Fourier Transform

GMRF: Gaussian Markov Random Field

IFFT: Inverse Fast Fourier Transform

IGO: Information Geometric Optimization

i.i.d.: independent identically distributed

MIMO: Multiple Input Multiple Output

ODE: Ordinary Differential Equation

PSR: Population Success Rule

r.h.s.: right hand side

RT: runtime

TPA: Two Point Adaptation

w.l.o.g.: without loss of generality

w.r.t.: with respect to

$\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$: sets of natural, integer, real and complex numbers respectively

$\mathbb{R}^n$: n-dimensional Euclidean space

$\mathcal{N}(\mathbf{m}, \mathbf{C})$: (Multivariate) Normal distribution with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{C}$

$\sim$: distributed as

$\mathbf{x}^T$: transpose of $\mathbf{x}$

$|x|$: absolute value or modulus of x, if x is a real or complex number respectively

$||\mathbf{x}||_2$: Euclidean norm of $\mathbf{x}$

$||\mathbf{X}||_F$: Frobenius norm of $\mathbf{X}$

$\mathbb{E}$: expected value

$\nabla_{\mathbf{x}}$: gradient with respect to $\mathbf{x}$

$\tilde{\nabla}_\theta$: natural gradient associated to a parameterization represented by θ

$\frac{\partial}{\partial x_i}$: partial derivative with respect to x_i

$\ln, \log$: natural logarithm

$\log_{10}$: logarithm with base 10

$\mathcal{O}(\cdot), \Theta(\cdot), o(\cdot)$: asymptotic Bachmann-Landau notation

$\circ$: composition of functions

$\lfloor a \rfloor$: largest integer $b \leq a$

$\lceil a \rceil$: smallest integer $b \geq a$

$\mathcal{N}_f$: invariant subspace of a function f

$||\mathbf{X}||_1$: sum of absolute values of entries of the matrix $\mathbf{X}$

$||\mathbf{X}^-||_1$: sum of absolute values of off-diagonal entries of the matrix $\mathbf{X}$

diag$\mathbf{X}$: diagonal matrix with diagonal entries those of $\mathbf{X}$

$\det \mathbf{X}$: determinant of $\mathbf{X}$

$\mathrm{Tr}(\mathbf{X})$: trace of $\mathbf{X}$

Card$(\mathbf{X})$: number of nonzero entries of $\mathbf{X}$

SO(k): special orthogonal group in dimension k

$\mathcal{S}^n$: set of symmetric matrices in $\mathbb{R}^{n \times n}$

$\mathcal{S}^n_{++}$: set of symmetric positive definite matrices in $\mathbb{R}^{n \times n}$

$||\mathbf{M}||$: operator norm of matrix $\mathbf{M}$

$\lambda_j(\mathbf{M})$: set of eigenvalues of matrix $\mathbf{M}$ indexed by j

$\mathbb{1}$: indicator

$\mathbf{1}$: vector with coordinates equal to 1

D_{KL}: Kullback-Leibler divergence

$e^{j\phi}$: polar form of a unit-length complex number with principal argument value ϕ

$\bar{z}$: complex conjugate of z

Arg$z$: principal argument of a complex nonzero number z

$\sup A$: supremum of set A

Chapter 1

Introduction

Gradient-free (or black-box) continuous optimization algorithms aim at optimizing an objective, or fitness, function f over a domain $\mathcal{D} \subset \mathbb{R}^n$ without using any information of the gradient of f. We refer to the domain $\mathcal{D}$ as the search space, and to its image by f as the objective space. Such problems arise often in practice, for example in simulators where the objective function is viewed as a black-box, thus the gradient of f cannot be computed, or it is costly to approximate, or even it does not exist. In fact, such methods are oriented in optimizing *difficult* (non-smooth, non-convex, rugged) objective functions, and to do so, they only use information of f-values on candidate solutions.

Several local algorithms, either deterministic or stochastic, have been developed to solve black-box problems in continuous domain [26]. The class of deterministic methods can be roughly classified in direct search methods and model-based methods [80] (even though hybrid methods of these classes also exist [53]). The former class contains simplicial methods such as the popular Nelder-Mead's simplex algorithm [66], generalized pattern search methods (GPS) introduced and analyzed by Torczon [94], or Mesh Adaptive Direct Search (MADS) methods [11, 3], which consist further generalizations of GPS. Model-based methods build a surrogate of the objective function for their updates, involving trust region methods that build the surrogate via polynomial interpolation [77, 74, 75, 24, 25], interpolation on radial basis functions [73, 35, 46], implicit filtering [33] and others.

Stochastic search (or randomized) methods on the other part perform the search by generating random candidate solutions in the search space, where the objective function is evaluated. Such approaches involve the pure random search, the simplest stochastic algorithm which generates the samples from a non-adaptive distribution, as well as Nesterov's random search [67], Particle Swarm Optimization [49, 71], the broad class of Evolutionary Algorithms [102] and others. Evolution Strategies, in particular, fall in the latter category and they are adaptive randomized methods which typically generate the candidate solutions as samples from a particular search distribution, which is adapted during the optimization process in order to achieve convergence. In this thesis we focus on methods based on the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [37], a randomized algorithm which performs the search with an adaptive multivariate normal distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C})$, recognised as a state-of-the-art

method among Evolution Strategies.

At each iteration t of this algorithm, solutions from a set of sampled vectors $\mathbf{x}_i \sim \mathcal{N}(\mathbf{m}_t, {\sigma_t}^2 \mathbf{C}_t)$ are ranked according to their corresponding f-values, and the parameters of the distribution, i.e. the mean vector $\mathbf{m}_t$, the step size σ_t and the covariance matrix $\mathbf{C}_t$ are updated, in order to generate *better* samples in terms of f-values at the following iteration. The update mechanism of CMA-ES, which is discussed in the following, is designed such that the method has many invariance properties that in turn provide a very efficient performance in addressing difficulties of the optimization process, such as non-separability and ill-conditioning.

One limitation of the method is its scaling with increasing dimensions. The fact that the number of adapted parameters is quadratic with the dimension n imposes learning rates of the distribution of the order of $\Theta(1/n^2)$ for a stable behaviour. Therefore a considerable amount of research has been focused on proposing ways to improve its efficiency for large scale optimization.

One of the primary objectives of this thesis is to analyze such methods and propose new paths of improving the efficiency of CMA-ES in large scale problems. In particular, we investigate three approaches for the latter purpose. The thorough analysis of large scale CMA-ES variants requires also their experimental evaluation. Therefore, a significant amount of the thesis work was focused on various aspects related to benchmarking such as the development of a large scale suite, the performance data collection of several solvers and their comparison and evaluation. Additionally, we focus on applications related to Radar problems: we investigate their proper modelling as optimization problems and we exploit information obtained from the benchmarking process in order to use appropriate methods for addressing them.

The rest of the thesis is organized as follows: in Chapter 2 we describe the CMA-ES algorithm and some of its most promising large scale variants. A study of the underlying ideas of these methods combined with their experimental evaluation and comparison is included. In Chapter 3 we introduce novel methods which employ tools of high dimensional estimation within the CMA-ES context, aiming at the improvement of the adaptation speed and thus the convergence speed on partially separable problems. Chapter 4 describes the Comparing Continuous Optimizers benchmarking platform [43, 44] (COCO) along with its methodology of algorithms' evaluation and introduces the `bbob-largescale` test suite in its finalised form, a suite with test problems of increased dimensions and low evaluation cost, that extends the original `bbob` suite [38] of COCO. Chapter 5 is dedicated to the Phase Code optimization and the Phased-Array Pattern design problems, two Radar applications for which we present the problem formulation, the methodology for their solution and the obtained results using CMA-ES-based methods.

Chapter 2

Black-Box Optimization and CMA-ES

This chapter starts by describing the fundamentals of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [37] and its relation to the Information Geometric Optimization algorithm [69]. A study of promising large scale variants of CMA-ES [97] is furthermore included, succeeded by a short description of the more recent dd-CMA-ES algorithm [7].

2.1 The $(\mu/\mu_w, \lambda)$-CMA-ES

As mentioned in the introduction, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) addresses problems of the form

$$\min_{\mathbf{x}\in\mathcal{D}\subset\mathbb{R}^n} f(\mathbf{x}) \tag{2.1}$$

by performing random search with an adaptive normal distribution $\mathcal{N}(\mathbf{m}, \sigma^2\mathbf{C})$. A population of λ candidate solutions $\mathbf{x}_i, i = 1, \ldots, \lambda$ is sampled from $\mathcal{N}(\mathbf{m}_t, {\sigma_t}^2\mathbf{C}_t)$ at each iteration t, and the distribution parameters are updated using techniques such as rank-based selection, recombination and cumulation described in the following.

Let $\{\mathbf{x}_{i:\lambda} : i = 1, \ldots, \lambda\}$ represent the population sorted according to increasing values of the objective function, i.e. $f(\mathbf{x}_{1:\lambda}) \leq \cdots \leq f(\mathbf{x}_{\lambda:\lambda})$. The updated mean vector is

$$\mathbf{m}_{t+1} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda} \tag{2.2}$$

where the coefficients w_i, called the recombination weights, are chosen such that $w_1 \geq \cdots \geq w_\mu > 0$ with $\mu \leq \lambda, \mu \in \mathbb{N}$. This procedure, called rank-based selection and recombination, favors the μ best solutions among the population and provides a better estimation $\mathbf{m}_{t+1}$ of the optimum of f. An important note here is that only the ranking of the candidate solutions is affecting the mean update, and not the actual $f-$values. This also holds for the rest of the parameter update rules in CMA-ES, making the algorithm invariant to compositions of the objective function with

strictly increasing transformations.

The covariance matrix is updated as:

$$\mathbf{C}_{t+1} = (1 - c_1 - c_\mu \sum_{i=1}^{\mu} w_i)\mathbf{C}_t + c_1 \mathbf{p}_t^c \mathbf{p}_t^{cT} + c_\mu \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda} \mathbf{y}_{i:\lambda}^T \tag{2.3}$$

where $\mathbf{y}_{i:\lambda} = \frac{\mathbf{x}_{i:\lambda} - \mathbf{m}_t}{\sigma_t}$. This shows that for a better covariance estimation, CMA-ES performs recombination of selected *steps* $\mathbf{y}_{i:\lambda}$. The update terms of the above expression can be interpreted in a different manner. The term $\sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda} \mathbf{y}_{i:\lambda}^T$ (which is of rank μ a.s., if the recombination weights are nonzero) is employed to estimate a better distribution using information only from the current iteration, while the rank-one update term $\mathbf{p}_t^c \mathbf{p}_t^{cT}$ cumulates information of successive mean steps from previous iterations [36]: the vector $\mathbf{p}_t^c$, also called the evolution path for the covariance matrix update, is adapted as:

$$\mathbf{p}_{t+1}^c = (1 - c_c)\mathbf{p}_t^c + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \frac{\mathbf{m}_{t+1} - \mathbf{m}_t}{\sigma_t} \tag{2.4}$$

with c_c its learning rate constant and $\mu_{\text{eff}} = 1/\sum_{i=1}^{\mu} w_i^2$ (for more details on the rationale of the parameter selection of CMA-ES we refer to [36]).

In contrast to the mean update, the recombination for the covariance matrix update may use the whole sampled population employing also negative weights w_i for $i \in \{\mu+1, \ldots, \lambda\}$ (a modification that has been adopted within CMA-ES, called Active CMA-ES [47]), which are appropriately chosen such that the matrix remains positive definite. The rank-μ update of equation (2.3) is also closely related to the generic Information Geometric Optimization algorithm (applied to the family of normal distributions), that we shortly describe in the following section.

Finally, the step size by default is updated as

$$\sigma_{t+1} = \sigma_t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_t^\sigma\|_2}{\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\|\mathbf{x}\|_2} - 1\right)\right), \tag{2.5}$$

where c_σ and d_σ constitute constant parameters in CMA-ES. As for the rank-one update of the covariance matrix, an evolution path $\mathbf{p}_t^\sigma$ is used for adapting the step size σ, now updated as

$$\mathbf{p}_{t+1}^\sigma = (1 - c_\sigma)\mathbf{p}_t^\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}} \mathbf{C}_t^{-\frac{1}{2}} \frac{\mathbf{m}_{t+1} - \mathbf{m}_t}{\sigma_t} \tag{2.6}$$

where $\mathbf{C}_t^{-\frac{1}{2}} = \mathbf{B}_t \mathbf{D}_t^{-1} \mathbf{B}_t^T$, $\mathbf{B}_t, \mathbf{D}_t$ representing the eigendecomposition components of $\mathbf{C}_t$, i.e. $\mathbf{C}_t = \mathbf{B}_t \mathbf{D}_t^2 \mathbf{B}_t^T$. Under random selection, the evolution path for the step size control is distributed as $\mathbf{p}_t^\sigma \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ [36], and if selection biases its norm to be larger (smaller) than $\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\|\mathbf{x}\|_2$, then the step size increases (decreases). This technique is also referred as Cumulative Step-Size Adaptation (CSA) and as mentioned in the following, alternatives have been proposed, in particular for large scale methods.

2.1.1 The convex quadratic case

In order to better illustrate the behaviour of CMA-ES, we consider in this subsection the case of optimizing a quadratic form in $\mathbb{R}^n$:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x}, \tag{2.7}$$

where $\mathbf{H} \in \mathcal{S}^n$. If the Hessian matrix $\mathbf{H}$ is furthermore positive definite, making f convex, then the level sets of f $\mathcal{L}_c^f = \{\mathbf{z} \in \mathbb{R}^n : \mathbf{z}^T\mathbf{H}\mathbf{z} = c\}$ are ellipsoids with axes in the directions of the eigenvectors of $\mathbf{H}$ and with axes lengths determined by the corresponding eigenvalues. The extreme eigenvalues determine the conditioning of the objective, quantified by the condition number

$$c = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \tag{2.8}$$

and ill-conditioning ($c \gg 1$) is a common encountered difficulty.

The typical behaviour of CMA-ES in such problems is that the search distribution learns these distinct axes in the sense that the iso-density sets $\mathcal{L}_c = \{\mathbf{z} \in \mathbb{R}^n : \mathbf{z}^T\mathbf{C}_t^{-1}\mathbf{z} = c\}$ of the centered distribution $\mathcal{N}(\mathbf{0}, \mathbf{C}_t)$ become aligned with the level sets of f and the inverse covariance matrix $\mathbf{C}_t^{-1}$ becomes proportional to $\mathbf{H}$. As a result, after the $\mathbf{C}$ adaptation, an originally ill-conditioned convex quadratic function can be optimized with the same convergence rate as the Sphere function $f_{\text{sphere}}(\mathbf{x}) = ||\mathbf{x}||^2$. Figure 2.1 illustrates the behaviour of the algorithm on f_{sphere}, as well as on the separable Ellipsoid function $f_{\text{elli}}(\mathbf{x}) = \sum_{i=1}^n 10^{6\frac{i-1}{n-1}} x_i^2$ and the rotated Ellipsoid function $f_{\text{ellirot}}(\mathbf{x}) = f_{\text{elli}}(\mathbf{Q}\mathbf{x}), \mathbf{Q} \in \text{SO}(n)$, with n distinct axes lengths. The latter two functions have the same topography of level sets, but for f_{elli} their axes have the directions of the standard basis of $\mathbb{R}^n$, while for f_{ellirot} they are rotated.

2.2 Information Geometric Optimization

The Information Geometric Optimization (IGO) method, established by Ollivier et al. [69], provides a generic framework for randomized adaptive optimization algorithms, often used in black-box problems. It considers algorithms which perform stochastic search with an adaptive distribution that belongs to a given family of distributions, parameterized by a set of parameters Θ. The IGO algorithm uses an adaptive transformation which maps the original objective function that we aim to optimize to a function over the parameter space Θ, and conducts the natural gradient ascent in the parameter space Θ of the mapped function.

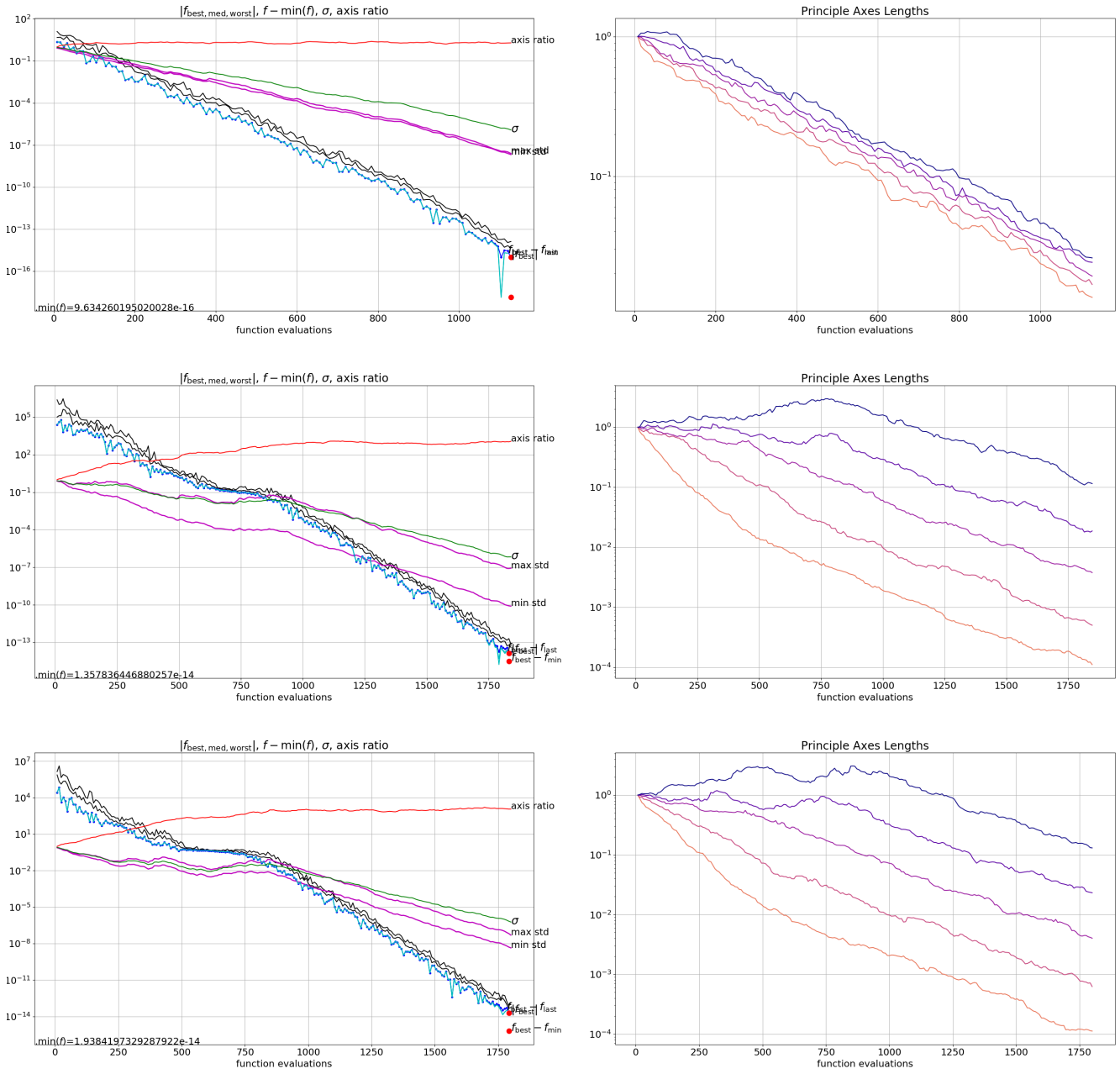

Figure 2.1: Single runs of CMA-ES on f_{sphere} (top), f_{elli} (middle) and f_{ellirot} (bottom) in dimension $n = 5$. The right column depicts the axes lengths of the search distribution ellipsoid. After the adaptation of the covariance matrix on f_{elli} and f_{ellirot} (i.e. approximately after 800 function evaluations), the step size σ decreases linearly in logarithmic scale.

2.2.1 Natural Gradient

Let us consider a (parameterized) family of probability distributions with a corresponding parameter set Θ and densities $P_\theta : \mathbf{X} \mapsto \mathbb{R}$, as well as a smooth function $g : \Theta \mapsto \mathbb{R}$. [1] The natural gradient of g is then defined as

$$\tilde{\nabla}_\theta g = \mathbf{I}^{-1} \frac{\partial g}{\partial \theta} \tag{2.9}$$

where $\mathbf{I}$ is the Fisher information matrix, i.e.

$$\mathbf{I}_{ij} = \int_{\mathbf{X}} \frac{\partial \ln P_\theta(\mathbf{x})}{\partial \theta_i} \frac{\partial \ln P_\theta(\mathbf{x})}{\partial \theta_j} P_\theta(d\mathbf{x}). \tag{2.10}$$

The Fisher matrix induces a Riemannian metric on Θ and the natural gradient of g (which is invariant of the chosen parametrization θ of P_θ) has the direction of the steepest ascent of g w.r.t. this metric [69].

2.2.2 Adaptive transformation of the objective function - IGO flow

Returning to the original problem (2.1), one has to transform the objective f to a function over the parameter space Θ. A possible choice for this would be to consider the expected fitness $\mathbb{E}_{P_\theta} f$ under the current parametrization and perform a step in the natural gradient direction of $-\mathbb{E}_{P_\theta} f$ (with the opposite sign since we consider the ascent direction), such that the θ adaptation concentrates the search distribution at the optimum of f. This technique is used in the class of Natural Evolution Strategies [101]. Instead, IGO utilizes $\mathbb{E}_{P_\theta} W^f_{\theta_t}$ where $W^f_{\theta_t}$ is a quantile-based rewriting of f, which aims to represent the weighted selection and recombination mechanisms described in section 2.1 and which, in contrast to $-\mathbb{E}_{P_\theta} f$, preserves the invariance property of IGO to compositions of f with strictly increasing transformations. For a detailed definition we refer to [69].

Then, the IGO flow is described by the ODE [69]:

$$\frac{d\theta_t}{dt} = \left(\tilde{\nabla}_\theta \int_{\mathbf{X}} W^f_{\theta_t}(\mathbf{x}) P_\theta(\mathbf{x}) d\mathbf{x} \right)_{|\theta=\theta_t} \tag{2.11}$$

$$= \mathbf{I}^{-1}(\theta_t) \int_{\mathbf{X}} W^f_{\theta_t}(\mathbf{x}) \frac{\partial \ln P_\theta(\mathbf{x})}{\partial \theta}_{|\theta=\theta_t} P_{\theta_t}(d\mathbf{x}). \tag{2.12}$$

After discretization, the IGO algorithm in iteration $t+1$ updates the parameter θ_t by a step in the direction of the natural gradient of $\mathbb{E}_{P_\theta} W^f_{\theta_t}$ at θ_t, that is:

$$\theta_{t+\delta t} = \theta_t + \delta t \mathbf{I}^{-1}(\theta_t) \int_{\mathbf{X}} W^f_{\theta_t}(\mathbf{x}) \frac{\partial \ln P_\theta(\mathbf{x})}{\partial \theta}_{|\theta=\theta_t} P_{\theta_t}(d\mathbf{x}). \tag{2.13}$$

In practice, the integral at the r.h.s. of equation (2.13) is approximated as a finite sum using the sampled popu-

[1] We assume in this section that necessary regularity conditions are met in all presented definitions. In particular, for multivariate normal distributions this assumption holds.

lation after rank-based selection and weighted recombination, as described in section 2.1. A result due to Akimoto et al. [8] states that (disregarding the rank one update), the covariance matrix update (2.3) is a particular instance of the IGO step when the search distribution family is the multivariate normal. From this viewpoint, the *small* step length δt in the natural gradient direction corresponds to the learning rates c_1, c_μ of the covariance matrix , which by default are of the order of $\Theta(1/n^2)$ in CMA-ES. As we see in the following chapter, large scale variants of CMA-ES attempt to increase these learning rates for a faster adaptation of the distribution.

2.3 A Comparative Study of Large-scale Variants of CMA-ES[2]

Apart from the learning rate setting of CMA-ES discussed above, the intrinsic complexity of the method in terms of memory and internal computational effort is quadratic in the dimensionality, n, of the black-box objective function to be solved. This complexity restricts its application when the number n of variables is in the order of a few hundred. For this reason, different "large"-scale variants of CMA-ES have been introduced over the past ten years. They all aim at a sub-quadratic space and time complexity [81, 5, 58, 59, 56, 50, 86]. The common feature of the variants is to restrict the model of the covariance matrix and provide a sparse representation that can be stored, sampled and updated in $\mathcal{O}(n \times m)$ operations with $m \ll n$. Yet the approaches to do so are quite different. On the one-hand, the seminal limited memory BFGS, L-BFGS [57], inspired the introduction of the limited memory CMA (LM-CMA, [58, 59]) where the main idea is to approximate at iteration $t \gg m$ the sum over t terms composing the covariance matrix by a sum over m terms. This same approach is used in the RmES algorithm [56]. On the other-hand, the sep-CMA [81] and VkD-CMA [5] algorithms enforce a predefined structure of the covariance matrix (for instance diagonal for the sep-CMA) and project at each iteration the updated matrix onto the restricted space.

This section presents a comparative review and performance assessment of the currently most promising large-scale variants of CMA-ES and their comparison to the well established L-BFGS algorithm. The review has been performed using the COCO benchmarking platform. We thoroughly describe the characteristics of the platform and of the `bbob-largescale` test suite, used for this comparative study, in chapter 4. Besides the general performance quantification and comparison, the benchmarking allows to identify defects of the algorithms or of their implementations (that shall be fixed in the near future).

2.3.1 Large-scale CMA-ES variants

We hereby present an overview of large-scale variants that have been introduced in recent years, with emphasis on the variants that are later empirically investigated.

[2]This study is based on the article "A Comparative Study of Large-scale Variants of CMA-ES" [97] by K. Varelas, A. Auger, D. Brockhoff, N. Hansen, O. Ait ElHara, Y. Semet, R. Kassab and F. Barbaresco, presented to the 2018 "Parallel Problem Solving from Nature" conference.

Cholesky-CMA. The sampling of the candidate solutions $(\mathbf{x}_t^i)_{1\le i\le\lambda}$ in CMA-ES is typically done by computing the eigendecomposition of the covariance matrix as $\mathbf{C}_t = \mathbf{B}_t\mathbf{D}_t^2\mathbf{B}_t^\top$ where $\mathbf{B}_t$ contains an orthonormal basis of eigenvectors, and $\mathbf{D}_t$ is a diagonal matrix containing the square roots of the corresponding eigenvalues. The square root of $\mathbf{C}_t$ is computed as $\mathbf{C}_t^{1/2} = \mathbf{B}_t\mathbf{D}_t\mathbf{B}_t^\top$ and used for sampling the candidate solutions as $\mathbf{x}_t^i = \mathbf{m}_t + \sigma_t\mathbf{C}_t^{1/2}\mathbf{z}_t^i$ with $\mathbf{z}_t^i \sim \mathcal{N}(0,\mathbf{I})$, where $\mathcal{N}(0,\mathbf{I})$ denotes a multivariate normal distribution with mean zero and covariance matrix identity. The eigendecomposition has a complexity of $\mathcal{O}\left(n^3\right)$ but is done only every $\mathcal{O}(n)$ evaluations (*lazy*-update) reducing the complexity of the sampling to $\mathcal{O}(n^2)$.

An alternative was proposed in [87]. Instead of using the eigendecomposition of the covariance matrix to sample candidate solutions, it uses a decomposition of $\mathbf{C}_t$ as $\mathbf{C}_t = \mathbf{A}_t\mathbf{A}_t^\top$. Indeed assume that $\mathbf{A}_t$ is known, then sampling $\mathbf{x}_i^t$ as $\mathbf{m}_t + \sigma_t\mathbf{A}_t\mathbf{z}_t^i$ with $\mathbf{z}_t^i \sim \mathcal{N}(0,\mathbf{I})$ results in a vector following $\mathcal{N}(\mathbf{m}_t, \sigma_t^2\mathbf{C}_t)$. When $\mathbf{A}_t$ is lower (or upper) triangular the decomposition is unique and called Cholesky factorization. However, in [87] the term Cholesky factorization is used without assuming that the matrix $\mathbf{A}_t$ is triangular. We will continue to use Cholesky-CMA for the ensuing algorithm to be consistent with the previous algorithm name.

The key idea for the Cholesky-CMA is that instead of adapting the covariance matrix $\mathbf{C}_t$, the Cholesky factor $\mathbf{A}_t$ is directly updated (and hence sampling does not require factorization of a matrix). The method solely conducts the rank-one update of the covariance matrix, $\mathbf{C}_{t+1} = (1-c_1)\,\mathbf{C}_t + c_1\mathbf{p}_{t+1}^c[\mathbf{p}_{t+1}^c]^\top$, by updating the matrix $\mathbf{A}_t$ such that $\mathbf{C}_{t+1} = \mathbf{A}_{t+1}\mathbf{A}_{t+1}^\top$. Indeed, let $\mathbf{v}_{t+1}$ be defined implicitly via $\mathbf{A}_t\mathbf{v}_{t+1} = \mathbf{p}_{t+1}^c$, then the update of $\mathbf{A}_t$ reads

$$\mathbf{A}_{t+1} = \sqrt{1-c_1}\mathbf{A}_t + \frac{\sqrt{1-c_1}}{\|\mathbf{v}_{t+1}\|_2^2}\left(\sqrt{1+\frac{c_1}{1-c_1}\|\mathbf{v}_{t+1}\|_2^2}-1\right)\mathbf{p}_{t+1}^c\mathbf{v}_{t+1}{}^\top, \tag{2.14}$$

if $\mathbf{v}_{t+1} \neq \mathbf{0}$ and $\mathbf{A}_{t+1} = \sqrt{1-c_1}\mathbf{A}_t$ if $\mathbf{v}_{t+1} = \mathbf{0}$ (see [87, Theorem 1]). A similar expression holds for the inverse $\mathbf{A}_{t+1}^{-1}$ (see [87, Theorem 2]). Sampling of a multivariate normal distribution using the Cholesky factor still requires $\mathcal{O}(n^2)$ operations due to the matrix-vector multiplication. However, the Cholesky-CMA has been used as foundation to construct numerically more efficient algorithms as outlined below. Recently, a version of CMA using Cholesky factorization enforcing triangular shapes for the Cholesky factors has been proposed [52].

Large-scale variants of CMA-ES The quadratic time and space complexity of CMA-ES (both the original and Cholesky variant) becomes critical with increasing dimension. This has motivated the development of large-scale variants with less rich covariance models, i.e., with $o(n^2)$ parameters. Reducing the number of parameters reduces the memory requirements and, usually, the internal computational effort, because fewer parameters must be updated. It also has the advantage that learning rates can be increased. Hence, learning of parameters can be achieved in fewer *number of evaluations*. Given the model is still rich enough for the problem at hand, this further reduces the computational costs to solve it in particular even when the f-computation dominates the overall costs. Hence, in the best case scenario, reducing the number of parameters from n^2 to n reduces the time complexity to solve the problem from n^2 to n if f-computations dominate the computational costs and from n^4 to n^2 if internal

computations dominate.

We review a few large-scale variants focussing on those benchmarked in the following.

sep-CMA-ES [81]. The separable CMA-ES restricts the full covariance matrix to a diagonal one and thus has a linear number of parameters to be learned. It loses the ability of learning the dependencies between decision variables but allows to exploit problem separability. The sep-CMA-ES achieves linear space and time complexity.

VkD-CMA-ES [5, 6]. A richer model of the covariance matrix is used in the VkD-CMA-ES algorithm where the eligible covariance matrices are of the form $\mathbf{C}_t = \mathbf{D}_t(\mathbf{I} + \mathbf{V}_t\mathbf{V}_t^\top)\mathbf{D}_t$ where $\mathbf{D}_t$ is a n-dimensional positive definite diagonal matrix and $\mathbf{V}_t = [\mathbf{v}_t^1 \dots \mathbf{v}_t^k]$ where $\mathbf{v}_t^i \in \mathbb{R}^n$ are orthogonal vectors [5]. The parameter k ranges from 0 to $n-1$: when $k = 0$ the method recovers the separable CMA-ES while for $k = n-1$ it recovers the (full)-CMA-ES algorithm. The elements of $\mathbf{C}_{t+1}$ are determined by projecting the covariance matrix updated by CMA-ES given in (2.3) denoted as $\hat{\mathbf{C}}_{t+1}$ onto the set of eligible matrices. This projection is done by approximating the solution of the problem $\underset{(\mathbf{D},\mathbf{V})}{\operatorname{argmin}} \|\mathbf{D}\,(\mathbf{I} + \mathbf{V}\mathbf{V}^\top)\,\mathbf{D} - \hat{\mathbf{C}}_{t+1}\|_F$ where $\|\cdot\|_F$ stands for the Frobenius norm. This projection can be computed without computing $\hat{\mathbf{C}}_{t+1}$. The space complexity of VkD-CMA-ES is $\mathcal{O}(nr)$ and the time complexity is $\mathcal{O}(nr \max(1, r/\lambda))$, where $r = k + \mu + \lambda + 1$. Note that the algorithm exploits both the rank-one and the rank-mu update of CMA-ES as the projected matrices result from the projection of the matrix $\hat{\mathbf{C}}_{t+1}$ updated with both terms.

A procedure for the online adaptation of k has been proposed in [6]. It tracks in particular how the condition number of the covariance matrix varies with changing k. The variant with the procedure of online adaptation of k as well as with fixed $k = 2$ is benchmarked in the following. The VkD-CMA algorithm uses *Two Point Adaptation* (TPA) to adapt the step-size. The TPA is based on the ranking difference between two symmetric points around the mean along the previous mean shift.

The limited-memory (LM) CMA [58, 59]. The LM-CMA is inspired by the gradient based limited memory BFGS method [57] and builds on the Cholesky CMA-ES. If $\mathbf{A}_0 = \mathbf{I}$, setting $a = \sqrt{1-c_1}$ and $b_t = \frac{\sqrt{1-c_1}}{\|\mathbf{v}_{t+1}\|_2^2}\left(\sqrt{1 + \frac{c_1}{1-c_1}\|\mathbf{v}_{t+1}\|_2^2} - 1\right)$, then (2.14) can be re-written as $\mathbf{A}_{t+1} = a^t\mathbf{I} + \sum_{i=1}^{t} a^{t-i} b_{i-1} \mathbf{p}_i^c \mathbf{v}_i^\top$. This latter equation is approximated by taking m elements in the sum instead of t. Initially, m was proposed to be fixed to $\mathcal{O}(\log(n))$. Later, better performance has been observed with m in the order of $\sqrt{n}$ [59], imposing $\mathcal{O}(n^{3/2})$ computational cost. Sampling can be done without explicitly computing $\mathbf{A}_{t+1}$ and the resulting algorithm has $\mathcal{O}(mn)$ time and space complexity. The choice of the m elements of the sum to approximate $\mathbf{A}_{t+1}$ seems to be essential. In L-BFGS the last m iterations are taken while for LM-CMA the backward $N_{\text{steps}} \times k$ iterations for $k = 0, \dots, m-1$ are considered (that is we consider the current iteration, the current iteration minus N_{steps} and so on). The parameter N_{steps} is typically equal to n. Since $\mathbf{A}_t\mathbf{v}_{t+1} = \mathbf{p}_{t+1}^c$, the inverse factor $\mathbf{A}_t^{-1}$ is employed for the computation of $\mathbf{v}_{t+1}$, but an explicit computation is not needed, similarly as for $\mathbf{A}_t$. To adapt the step-size, the LM-CMA uses the *population success rule* (PSR) [58].

A variant of LM-CMA was recently proposed, the LM-MA, which is however not tested here because (i) the code is not available online and (ii) the performance of LM-MA seems not to be superior to LM-CMA [60].

The RmES [56]. The idea for the RmES algorithm is similar to the LM-CMA algorithm. Yet, instead of using the Cholesky-factor, the update of $\mathbf{C}_t$ is considered. Similarly as for LM-CMA, if $\mathbf{C}_0 = \mathbf{I}$ and solely the rank-one update is used for CMA-ES we can write the update as $\mathbf{C}_t = (1-c_1)^m \mathbf{I} + c_1 \sum_{i=1}^{m} (1-c_1)^{m-i} \hat{\mathbf{p}}_i^c \hat{\mathbf{p}}_i^{c^\top}$. In RmES, m terms of the sum are considered and $m = 2$ is advocated. Additionally, like in LM-CMA, the choice of terms entering the sum is by maintaining a temporal distance between generations. Sampling of new solutions is done from the m vectors without computing the covariance matrix explicitly. The RmES adapts the step-size similarly to PSR.

A main difference to LM-CMA is that RmES is formulated directly on the covariance matrix, thus an inverse Cholesky factor is not needed. This does not improve the order of complexity, though, which is $\mathcal{O}(mn)$ as in LM-CMA.

The presented algorithms do not of course form an exhaustive list of proposed methods for large-scale black-box optimization. We refer to [60] for a more thorough state-of-the-art and point out that our choice is driven by variants that currently appear to be the most promising or by variants like sep-CMA, important to give baseline performance.

2.3.2 Experimental results

We assess the performance of implementations of the algorithms presented in the previous section on the `bbob-largescale` suite. We are particularly interested to identify the scaling of the methods, possible algorithm defects, and to quantify the impact of the population size. Because we benchmark algorithm *implementations*, as opposed to mathematical algorithms, observations may be specific to the investigated implementation only.

Experimental Setup. We run the algorithms sep-CMA, LM-CMA, VkD-CMA, RmES on the default `bbob` test suite in dimensions $2, 3, 5, 10$ and on the `bbob-largescale` suite implemented in COCO. Additionally, we run the limited memory BFGS, L-BFGS, still considered as the state-of-the-art algorithm for gradient based optimization [57]. Gradients are estimated via finite-differences.

For VkD-CMA, the Python implementation from `pycma`, version 2.7.0, was used, for sep-CMA the version from `sites.google.com/site/ecjlmcma`, and for L-BFGS the optimization toolbox of `scipy` 1.2.1. We consider two versions of LM-CMA provided by the author at `sites.google.com/site/ecjlmcma` and `.../lmcmaeses` related to the articles [58] denoted LM-CMA'14 and [59] denoted LM-CMA. The implementation of RmES was kindly provided by its authors [56].

Experiments were conducted with default[3] parameter values of each algorithm and a maximum budget of $5 \cdot 10^4 n$. Automatic restarts are conducted once a default stopping criterion is met until the maximum budget is reached. For each function, fifteen instances are presented. For the first run and for all (automatic) restarts, the initial point was uniform at random between $[-4, 4]^n$ for all algorithms, while the initial step-size was set to 2 for all CMA variants.

For LM-CMA, sep-CMA and RmES, population sizes of $4 + \lfloor 3 \log n \rfloor$, $2n + \lfloor 10/n \rfloor$ and $10n$ were tested and the

[3] Except L-BFGS, where the `ftol` parameter was set to the machine precision for very high accuracy.

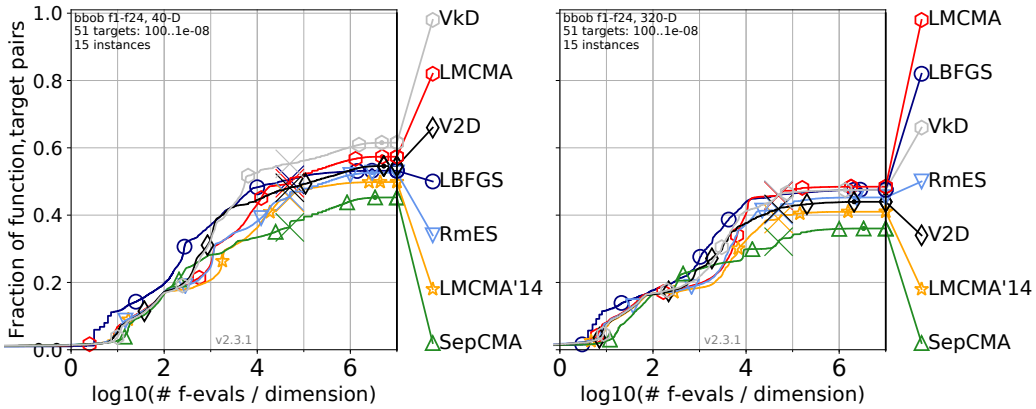

Figure 2.2: Bootstrapped ECDF of the number of objective function evaluations divided by dimension (FEvals/D) for 51 targets in $10^{[-8..2]}$ for all functions in 40-D (left) and 320-D.

experiments were conducted for the same budget and instances. A suffix P2 (P10) is used to denote the respective algorithms. For VkD-CMA, a second experiment has been run where the number of vectors was fixed to $k = 2$, denoted as V2D-CMA.

Performance assessment. We measure the number of function evaluations to reach a specified target function value, denoted as *runtime*, RT. The average runtime, aRT, for a single function and target value is computed as the sum of all evaluations in unsuccessful trials plus the sum of runtimes in all successful trials, both divided by the number of successful trials. For Empirical Cumulative Distribution Functions (ECDF) and in case of unsuccessful trials, runtimes are computed via simulated restarts [43, 44] (bootstrapped ECDF). The *success rate* is the fraction of solved problems (function-target pairs) under a given budget as denoted by the y-axis of ECDF graphs. Horizontal differences between ECDF graphs represent runtime ratios to solve the same respective fraction of problems (though not necessarily the same problems) and hence reveal how much faster or slower an algorithm is.

Overview. A complete presentation of the experimental results is available at `ppsndata.gforge.inria.fr`. Fig. 2.2 presents for each algorithm the runtime distribution aggregated over all functions. Overall, the distributions look surprisingly similar in particular in larger dimension. After $5 \cdot 10^4 n$ evaluations in 320-D, between 32% (sepCMA) and 46% (LMCMA) of all problems have been solved. In all dimensions, for a restricted range of budgets, the success rate of L-BFGS is superior to all CMA variants. The picture becomes more diverse with increasing budget where L-BFGS is outperformed by CMA variants. We emphasize that even domination over the entire ECDF does not mean that the algorithm is faster on every single problem, because runtimes are shown in increasing order *for each algorithm*, hence the order of problems as shown most likely differs.

Up to a budget of $10^4 n$, the performance similarity between LM-CMA and RmES is striking. The performance is almost identical on the Sphere, Ellipsoid, Linear Slope and Sum of Different Powers functions in dimensions equal or larger to 20. On the Bent Cigar function in dimensions greater or equal to 80 and for a budget larger than $10^4 n$, LM-CMA is notably superior to RmES.

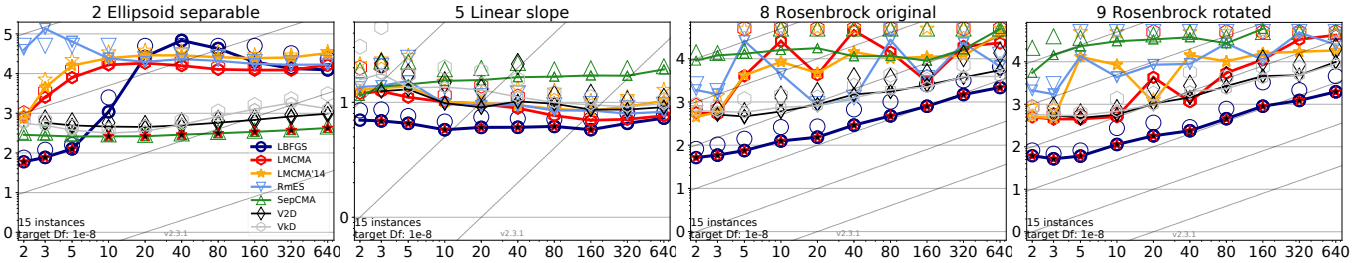

Figure 2.3: Scaling graphs: Average Runtime (aRT) divided by dimension to reach a target of 10^{-8} versus dimension for selected functions. Light symbols give the maximum number of evaluations from the longest trial divided by dimension.

Scaling with dimension. Fig. 2.3 shows the average runtime scaling with dimension on selected functions. On the separable Ellipsoid for $n \geq 20$ sep-CMA with population size $\geq 2n$ (not shown in Fig. 2.3) and VkD scale worse than linear. Starting from dimension 20, LM-CMA and RmES show runtimes of aRT ≈ 1.1–$4.0 \times 10^4 n$. With default population size, sep-CMA performs overall best and is for $n \geq 20$ even more than twenty times faster than L-BFGS. The latter scales roughly quadratically for small dimensions and (sub-)linear (with a much larger coefficient) for large dimensions. This behavior is a result of a transition when the dimension exceeds the rank (here 10) of the stored matrix. On the linear function, algorithms scale close to linear with a few exceptions. With population size $2n+\lfloor 10/n \rfloor$ or larger (not shown in Fig. 2.3), the scaling becomes worse in all cases (which means a constant number of iterations is not sufficient to solve the "linear" problem). In particular, sep-CMA reveals in this case a performance defect due to a diverging step-size (which disappears with option `'AdaptSigma': 'CMAAdaptSigmaTPA'`), as verified with single runs. On both Rosenbrock functions, L-BFGS scales roughly quadratically.

Restricting the model. The particular case of the ill-conditioned non-separable ellipsoidal function in Fig. 2.4 illustrates interesting results: in 20D, VkD-CMA solves the function, i.e. reaches the best target value faster (by a factor of 7 at least) than any other method. In 640-D any other CMA variant with default parameter values except sep-CMA outperforms it.

On the Ellipsoid function only VkD-CMA scales quadratically with the dimension. All other algorithms either scale linearly or do not solve the problem for larger dimension. On the Discus function (with a fixed proportion of short axes), VkD-CMA slows down before to reach the more difficult targets and exhausts the budget. An unusual observation is that LM-CMA performs considerably better on the Attractive Sector function in the smallest *and largest dimensions*. We do not see this effect on LM-CMA'14, where the choice of the number of the direction vectors is smaller and random. Thus, these effects indicate the importance of properly choosing m [58]. Even though the covariance matrix model provided by VkD-CMA is richer, the method is outperformed by RmES and LM-CMA, e.g. on the Ellipsoid function in dimension greater than 80 and on the Discus function for $n \geq 320$. This suggests that k is adapted to too large values thereby impeding the learning speed of the covariance matrix.

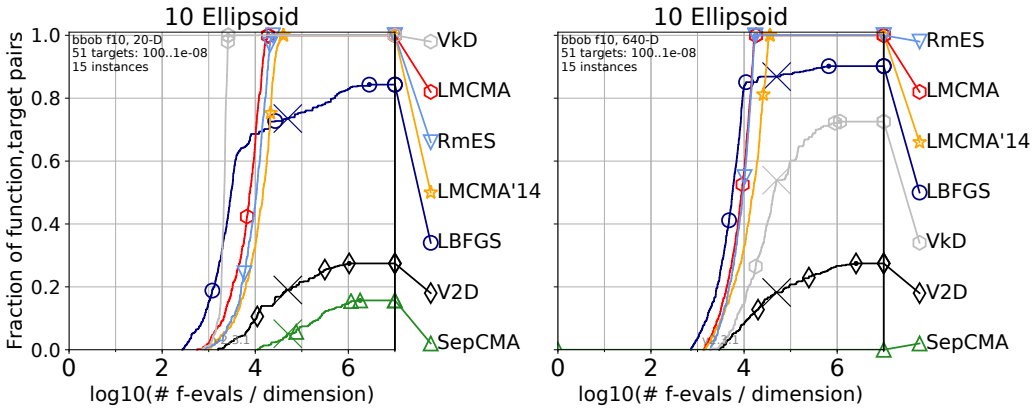

Figure 2.4: Bootstrapped ECDF of the number of objective function evaluations divided by dimension (FEvals/D) for 51 targets in $10^{[-8..2]}$ for the ellipsoid function in 20-D and 640-D.

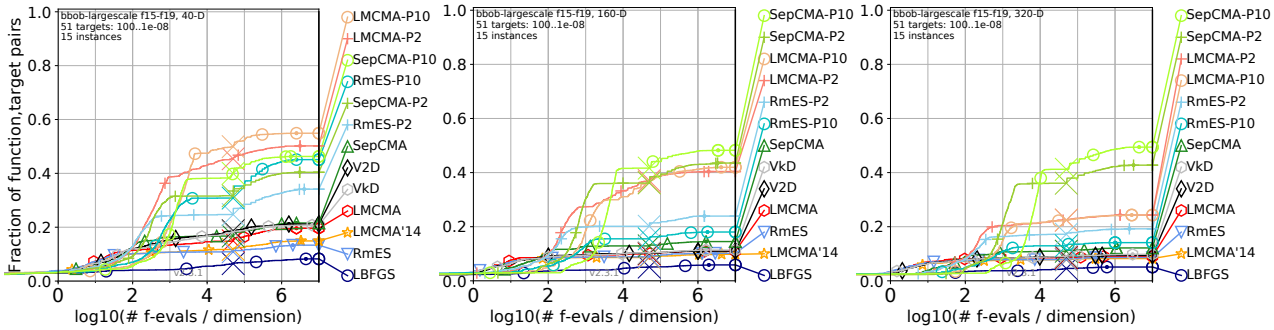

Figure 2.5: Bootstrapped ECDF of the number of objective function evaluations divided by dimension (FEvals/D) for 51 targets in $10^{[-8..2]}$ for the group of multimodal functions with adequate structure in 40-D (left), 160-D (middle) and 320-D (right).

Fixed versus adapted k. In order to investigate the effect of k-adaptation, we compare VkD-CMA with adaptive and fixed $k = 2$. Only in few cases the latter shows better performance. This is in particular true for the intrinsically not difficult to solve Attractive Sector function, indicating that the procedure of k adaptation could impose a defect.

Impact of population size. In Fig. 2.5, the effect of larger populations is illustrated for the multimodal functions with adequate global structure. The CMA variants with default population size and L-BFGS are clearly outperformed, solving less than half as many problems. That is, increased population size variants reach better solutions. Yet, the overall performance drops notably with increasing dimension. As expected, on the weakly-structured multimodal functions `f20-f24`, larger populations do not achieve similar performance improvements.

2.3.3 Discussion and Conclusion

We conclude in the presented review that in all dimensions, L-BFGS generally performs best with lower budgets and is outperformed by CMA variants as the budget increases. On multi-modal functions with global structure, CMA-ES variants with increased population size show the expected decisive advantage over L-BFGS. For larger dimension, the performance on these multi-modal functions is however still unsatisfying. The study has revealed

some potential defects of algorithms (k-adaptation in VkD-CMA on the Attractive Sector, Ellipsoid and Discus) and has confirmed the impact and criticality of the choice of the m parameter in LM-CMA. The VkD-CMA that appears to be a more principled approach and includes a diagonal component and the rank-μ update of the original CMA-ES, overall outperforms LM-CMA and RmES in smaller dimension, while LM-CMA overtakes for the large budgets in larger dimensions. On single functions, the picture is more diverse, suggesting possible room for improvement in limited memory and VkD-CMA approaches.

2.4 Diagonal acceleration

Apart from the methods analyzed above, another promising variant of CMA-ES, called dd-CMA-ES [7], has been proposed more recently. Within this method, a decomposition of the form $\mathbf{D}\tilde{\mathbf{C}}\mathbf{D}$ of the covariance matrix is considered, where $\mathbf{D}$ is diagonal and the terms $\mathbf{D}$, $\tilde{\mathbf{C}}$ are separately updated with different learning rates, which depend on the corresponding number of adapted parameters. In particular, the authors propose learning rates of the order $\Theta(n^{-3/4})$ and $\Theta(n^{-7/4})$ for $\mathbf{D}$ and $\tilde{\mathbf{C}}$ respectively. This way, the method attempts to exploit the separability property of an objective function, for which an axis parallel search distribution (represented by the diagonal $\mathbf{D}$ matrix) is sufficient to optimize, while it doesn't deteriorate the performance of CMA-ES on non-separable problems.

Note that the learning rate of the $\tilde{\mathbf{C}}$ component is less conservative than the default CMA-ES rate, thus even non-separable ill-conditioned problems can be solved more efficiently with dd-CMA-ES. A learning rate setting dependent on the number of adapted parameters will be useful in the methods that we propose, as we see in the following chapter.

Chapter 3

Novel approaches with high-dimensional estimation methods

In this chapter we introduce newly proposed algorithms which combine high-dimensional estimation tools with CMA-ES. As we previously saw, Separable CMA-ES restricts the search model to an axes-parallel distribution which in turn has a substantially good performance in separable problems, e.g. for functions $f : \mathbb{R}^n \mapsto \mathbb{R}$ which can be expressed as $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(x_i)$.

The methods proposed in this chapter have as common goal to accelerate the adaptation speed of the covariance matrix to a class of objective functions with sparsity properties which is broader than the class of separable functions.

3.1 Preliminairies and related work

We recall the definitions of an objective function's invariant subspace, and of the class of partially separable functions [68].

Definition 3.1.1 (Invariant Subspace)**.** The invariant subspace $\mathcal{N}_f$ of a function $\mathbb{R}^n \ni \mathbf{x} \mapsto f(\mathbf{x}) \in \mathbb{R}$ is the largest linear subspace of $\mathbb{R}^n$ such that for all $\mathbf{x} \in \mathbb{R}^n$ and for all $\mathbf{w} \in \mathcal{N}_f$ we have $f(\mathbf{x}) = f(\mathbf{x} + \mathbf{w})$.

f

One simple example is the function $f(\mathbf{x}) = x_1^2$. Its invariant subspace according to (3.1.1) is $\mathcal{N}_f = \{0\} \times \mathbb{R}^{n-1}$ with dimension $n-1$. Note that for $g(\mathbf{x}) = (\sum_{i=1}^{n} x_i)^2$, the invariant subspace $\mathcal{N}_g = \{\mathbf{z} \in \mathbb{R}^n : \mathbf{1}^T\mathbf{z} = 0\}$ is again of dimension $n-1$ and in contrast to the previous case the gradient and Hessian are fully dense.

Definition 3.1.2 (Partial Separability)**.** A function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is partially separable if it can be decomposed as a sum of elementary functions with large invariant subspaces, i.e. there exist $f_i : \mathbb{R}^n \mapsto \mathbb{R}$ such that $f = \sum_i f_i$ and the

corresponding spaces $\mathcal{N}_{f_i}$ have dimensions $n_i \gg 0$.

A further subclass of partially separable functions contains functions of the form $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$ where each elementary term f_i depends on few search space coordinates. In this case, the (vague) statement $n_i \gg 0$ of the definition 3.1.2 translates to the (also vague) statement that the Hessian of f is sparse, if f is twice differentiable. We attempt to propose methods which improve the performance of CMA-ES on this particular subclass of functions by performing the search with a search model whose precision matrix (i.e. inverse covariance matrix) matrix is sparse. Our goal is to reduce if possible the number of free parameters of the search model, thus increase the adaptation speed, without deteriorating the performance of CMA-ES. In the case of a smooth objective function f, this would mean to exploit sparsity of the Hessian. However, no assumption is imposed on the regularity of f. The assumption that such a search distribution is sufficient to optimize functions of the above form arises naturally, since empirical evidence indicates that in the case of optimizing a convex quadratic function with CMA-ES, the covariance matrix approximates its inverse Hessian (up to a scalar factor). This property allows, after adapting $\mathbf{C}$, to optimize any convex quadratic function with the same convergence rate as the Sphere function $f(\mathbf{x}) = \|\mathbf{x}\|_2^2$, and efficiently address ill-conditioned problems.

The challenge of estimating covariance or precision matrices when the dimension is large, due to the need of large numbers of samples and to the cumulation of significant amounts of estimation errors, leads to the need of discovering efficient high dimensional estimation methods and developing corresponding tools. Several approaches have been proposed, often with the assumption of sparsity properties of the matrix to be estimated, which include soft/hard thresholding, l_1 penalization, column-by-column estimation and others [29]. Among the first studies focusing on sparse precision matrix estimation is the seminal article of Dempster [28].

In the proposed methods of the following sections we employ Graphical Lasso regularization, hard thresholding and a greedy approach for Gaussian Markov Random Field Estimation.

3.1.1 Real-Valued GOMEA exploiting conditional linkage structure

Before introducing the novel approaches, it is useful to mention the Real-Valued Gene-Pool Optimal Mixing Evolutionary Algorithm (RV-GOMEA) with conditional linkage models [23], a related method that exploits a conditional dependency structure of a problem's search variables.

Originally GOMEA [93] was introduced as an optimization method over the discrete domain, having its roots at the Linkage Tree Genetic Algorithm (LTGA) [92]. The method attempted to exploit the linkage structure of a problem, which describes dependencies of search variables, and define the variation and recombination operators according to this structure. Naturally, such a method may be advantageous for Gray-Box Optimization (GBO) problems where the linkage structure is up to some extent known and allows for more efficient partial evaluations (i.e. function evaluations after varying few search coordinates) than the full re-evaluation of the objective function [22]. This is not

restrictive, though, since the problem structure may be learned via a statistical analysis of the sampled population.

The real valued GOMEA [22] is an extension of GOMEA to the continuous domain, that combines the linkage structure exploitation idea with aspects of the AMaLGaM algorithm [21]. The latter belongs to the family of Estimation of Distribution Algorithms, and the real valued GOMEA estimates a normal distribution for each group of dependent variables, in order to perform the variation of solutions, succeeded by their recombination. Initially the method was evaluated with the so-called marginal product linkage structure (with mutually exclusive groups of dependent variables) and the linkage tree structure, also used in [92]. More recently, the authors proposed ways to use conditional linkage models in RV-GOMEA, which in turn improved substantially the performance and scalability [23].

3.2 Sparse Inverse Covariance Learning for CMA-ES with Graphical Lasso[1]

The Graphical Lasso [31, 27] was introduced to estimate distributions with a sparse precision, i.e. inverse covariance, matrix. With this property, one introduces parametric models with conditionally independent search coordinates, a procedure also known as covariance selection [28]. In particular, if $\mathbf{\Sigma}$ is the sample estimation of a covariance matrix, the solution of

$$\underset{\mathbf{X}\epsilon\mathcal{S}^n_{++}}{\text{minimize}}\, \text{tr}(\mathbf{\Sigma X}) - \log\det \mathbf{X} + \alpha\|\mathbf{X}\|_1 \tag{3.1}$$

provides the sparse model estimation, where $\mathbf{X}$ represents the precision matrix to be estimated, $\mathcal{S}^n_{++}$ is the set of symmetric positive definite $n\times n$ matrices and the penalty factor α controls the tradeoff between the log-likelihood and the penalization term $\|\mathbf{X}\|_1 = \sum^n_{i,j=1}|\mathbf{X}_{ij}|$. For $\alpha = 0$, the solution $\mathbf{X}^*$ of (3.1) is $\mathbf{X}^* = \mathbf{\Sigma}^{-1}$, since the Kullback-Leibler divergence of the distributions parameterized by $\mathbf{X}$ and $\mathbf{\Sigma}$ is decomposed as:

$$\begin{aligned} D_{\text{KL}}(\mathcal{N}(\mathbf{0},\mathbf{\Sigma})\|\mathcal{N}(\mathbf{0},\mathbf{X}^{-1})) &= \frac{1}{2}\left(\text{tr}(\mathbf{\Sigma X}) - n + \log\left(\frac{\det \mathbf{X}^{-1}}{\det \mathbf{\Sigma}}\right)\right) \\ &= \frac{1}{2}\left(\text{tr}(\mathbf{\Sigma X}) - \log\det \mathbf{X}\right) - \frac{1}{2}\left(n + \log\det \mathbf{\Sigma}\right). \end{aligned} \tag{3.2}$$

The l_1 penalization in (3.1) is employed to force sparsity on the precision matrix $\mathbf{X}$, or equivalently on the absolute partial correlation matrix $|\text{diag}\mathbf{X}^{-1/2}\mathbf{X}\text{diag}\mathbf{X}^{-1/2}|$, and can be viewed as a convex relaxation of the number of non zero entries of $\mathbf{X}$, **Card**$(\mathbf{X})$ (which makes (3.1) a NP-hard problem [27]).

In a black-box scenario where the sparsity structure is unknown, estimating the precision matrix with the Graphical Lasso serves exactly the purpose of discovering this structure. In the context of CMA-ES, in order to learn sparse search models, the candidate solutions are generated from the regularized distribution that solves (3.1) and the original update rules are used. In practice, the Graphical Lasso is applied to standardized variables, thus when

[1] This section is based on the article "Sparse Inverse Covariance Learning for CMA-ES with Graphical Lasso" [98] by K. Varelas, A. Auger and N. Hansen, presented to the 2020 "Parallel Problem Solving from Nature" conference.

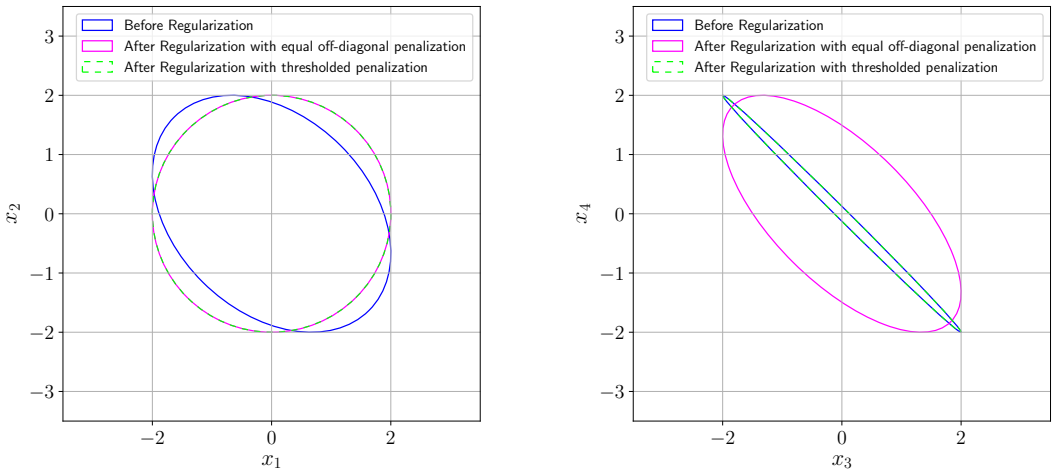

Figure 3.1: Equal-density ellipses of the marginal distributions of (x_1, x_2) and (x_3, x_4) before and after regularization with off-diagonal only and with thresholded penalty factors. The random vector $\mathbf{x} = (x_1, \ldots, x_4)$ is distributed as $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$ where $\mathbf{C}$ is a block-diagonal covariance matrix of the form $\mathbf{C} = \left(\begin{smallmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_2 \end{smallmatrix}\right), \mathbf{C}_1, \mathbf{C}_2$ being of size 2×2. The solid magenta line illustrates the effect of regularization when only off-diagonal elements are penalized with the same factor α. The factor value is chosen as the minimal value that achieves an isotropic distribution for the pair of weakly dependent variables (x_1, x_2) (left, with an axis ratio $\sqrt{2}$), i.e. $\alpha = 1/3$ in this particular example. The search distribution of the strongly dependent pair (x_3, x_4) (right, with an axis ratio $\sqrt{1000}$) is drastically affected. The dashed line (green) corresponds to thresholded regularization according to (3.3). In this case, only the precision matrix entry corresponding to the pair (x_1, x_2) is penalized, i.e. the chosen factors are: $\alpha_{ij} = 1/3$ if $(i, j) \in \{(1, 2), (2, 1)\}$ else $\alpha_{ij} = 0$.

solving (3.1) $\mathbf{\Sigma}$ is the correlation matrix provided by the CMA-ES update.

3.2.1 Equal weights and effect on conditioning

Problem (3.1) imposes the same penalization factor α on all precision entries, and the alternative regularization term $\alpha\|\mathbf{X}^-\|_1 = \alpha \sum_{i \neq j} |\mathbf{X}_{ij}|$, which penalizes only the off-diagonal entries, has been proposed e.g. in [63]. This kind of penalization leads to a consistent reduction of the axes length ratios learned by CMA-ES after the regularization step, as illustrated in Figure 3.1 for a 4 dimensional block diagonal case.

Recently, tools for an extension of (3.1) with non-equal penalization factors, i.e. for solving:

$$\underset{\mathbf{X} \in \mathcal{S}^n_{++}}{\text{minimize}} \operatorname{tr}(\mathbf{\Sigma}\mathbf{X}) - \log\det \mathbf{X} + \sum_{i,j=1}^{n} \alpha_{ij} |\mathbf{X}_{ij}| \tag{3.3}$$

with selected $\alpha_{ij} \geq 0$ have been developed [54]. In the following, this formulation is used along with a simple rule for selecting the penalty factors in order to surpass the above effect: precision entries are penalized only if the corresponding absolute partial correlations, i.e. the entries of $|\mathrm{diag}\mathbf{X}^{-1/2}\mathbf{X}\mathrm{diag}\mathbf{X}^{-1/2}|$, are below a threshold τ.

3.2.2 Algorithm

In this section we introduce the proposed algorithm, denoted as gl-CMA-ES. It only uses recombination with positive weights for the update of the covariance matrix, in order to ensure its positive definiteness. The differences with respect to the original CMA-ES setting with positive recombination weights [2] are highlighted in Algorithm 1, while Algorithm 2 describes the regularization step. The minimization step in line 6 of Algorithm 2 is solved using [54].

For reasons of stability, and since the number of degrees of freedom for the covariance matrix is $n(n+1)/2$, the corresponding learning rate in CMA-ES is of the order of $\mathcal{O}(1/n^2)$. In other large scale variants of CMA-ES, e.g. the Separable CMA-ES [81], the degrees of freedom of the search model are reduced and the adaptation is performed faster. Similarly, in our approach the learning rates depend on the number of non zero entries of the Lasso estimated precision matrix, ranging from $\mathcal{O}(1/n)$ for sparse to $\mathcal{O}(1/n^2)$ for dense matrices. Furthermore, limited memory methods have been proposed [50, 59], aiming at reducing the internal space and time complexity of CMA-ES. Such methods, though, do not exploit properties such as separability in order to accelerate the convergence [50, 97].

The algorithm coincides with CMA-ES if the threshold is chosen as $\tau = 0$, that is if the l_1 penalization is not applied. If this holds, the sampling matrix $\mathbf{C}_{t+1}^{\text{reg}}$ is equal to $\mathbf{C}_t$ in line 4 of Algorithm 1, thus the candidate solutions are sampled from $\mathcal{N}(\mathbf{m}_t, \sigma_t^2 \mathbf{C}_t)$, see lines 9 and 10. Additionally, the evolution path for the adaptation of the step size follows the same update rule as in CMA-ES, see line 14 of Algorithm 1. The learning rates c_1, c_μ are defined in a compatible way with CMA-ES in line 6, when the precision matrix is fully dense, i.e. when $n_z = n^2$.

Note that the invariance property to strictly monotonic transformations of the objective function f that CMA-ES possesses is maintained in the algorithm. However, invariance to affine transformations of the search space breaks when regularization is applied, i.e. when setting $\tau > 0$.

3.2.3 Results

We present experimental results on representative problems included in Table 3.1, in order to verify whether the proposed approach is able to identify the correct sparse structure of the objective function's Hessian matrix. All experiments were performed with an initial step size $\sigma_0 = 1$ and with a starting point $\mathbf{x}_0$ defined in Table 3.1.

We consider as a first test case the function f_{ellisub} which is constructed by composing the Ellipsoid function f_{elli} with a rotational search space transformation as defined in Table 3.1. This results in a non-separable ill-conditioned problem with maximal sparsity, since the upper triangular part of the Hessian of f_{ellisub} has exactly one non zero entry. Figure 3.2 presents the gain in convergence speed in terms of number of function evaluations of gl-CMA-ES compared to the default CMA-ES (with positive recombination weights). It also shows the performance scaling with dimension, compared to other large scale variants of CMA-ES, namely the Separable CMA-ES [81], the VkD-CMA-ES [6] and the dd-CMA-ES [7], as well as with the Active CMA-ES [47], i.e. the algorithm that uses the entire sample

[2] An extension of CMA-ES, called Active CMA-ES [47], that performs recombination with both positive and negative weights using all sampled solutions has been proposed.

Algorithm 1 gl-CMA-ES

Set parameters: $\lambda = 4 + \lfloor 3 \ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+\frac{1}{2})-\ln i}{\sum_{j=1}^{\mu} \ln(\mu+\frac{1}{2})-\ln j}$ for $i = 1 \dots \mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\mu_w+2}{n+\mu_w+3}$, $d_\sigma = 1 + 2\max\{0, \sqrt{\frac{\mu_w-1}{n+1}} - 1\} + c_\sigma$, $c_c = \frac{4+\mu_w/n}{n+4+2\mu_w/n}$,
Initialize: $\mathbf{p}_t^c \leftarrow \mathbf{0}$, $\mathbf{p}_t^\sigma \leftarrow \mathbf{0}$, $\mathbf{C}_t \leftarrow \mathbf{I}$, $t \leftarrow 0$, τ,
while termination criteria not met **do**
  $\mathbf{C}_{t+1}^{\text{reg}} \leftarrow \text{REGULARIZE}(\mathbf{C}_t, \tau)$,
  $n_z \leftarrow \#|\mathbf{C}_{t+1}^{\text{reg}\,-1}| > 0$,
  $c_1 \leftarrow \frac{2}{(n_z/n+1.3)(n+1.3)+\mu_w}, c_\mu \leftarrow \min\{1-c_1, 2\frac{\mu_w+1/\mu_w-1.75}{(n_z/n+2)(n+2)+\mu_w}\}$,

  for $k \leftarrow 1, \dots, \lambda$ **do**
    $\mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_{t+1}^{\text{reg}})$
    $\mathbf{x}_k \leftarrow \mathbf{m}_t + \sigma_t \mathbf{z}_k$
    $f_k \leftarrow f(\mathbf{x}_k)$
  end for
  $\mathbf{m}_{t+1} \leftarrow \sum_{k=1}^{\mu} w_k \mathbf{x}_{k:\mu}$
  $\mathbf{p}_{t+1}^\sigma \leftarrow (1-c_\sigma)\,\mathbf{p}_t^\sigma + \sqrt{c_\sigma(2-c_\sigma)\mu_w}\,\mathbf{C}_{t+1}^{\text{reg}\,-\frac{1}{2}} \frac{\mathbf{m}_{t+1}-\mathbf{m}_t}{\sigma_t}$,
  $h_\sigma \leftarrow \mathbb{1}_{\|\mathbf{p}_{t+1}^\sigma\|_2 < (1.4+\frac{2}{n+1})\sqrt{1-(1-c_\sigma)^{2(t+1)}}\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|_2}$,
  $\delta(h_\sigma) \leftarrow (1-h_\sigma)c_c(2-c_c)$,
  $\mathbf{p}_{t+1}^c \leftarrow (1-c_c)\,\mathbf{p}_t^c + h_\sigma\sqrt{c_c(2-c_c)\mu_w}\,\frac{\mathbf{m}_{t+1}-\mathbf{m}_t}{\sigma_t}$,
  $\mathbf{C}_{t+1}^\mu \leftarrow \sum_{k=1}^{\mu} w_k \mathbf{z}_{k:\mu}\mathbf{z}_{k:\mu}^T$,
  $\mathbf{C}_{t+1} \leftarrow (1 + c_1\delta(h_\sigma) - c_1 - c_\mu)\mathbf{C}_t + c_1\mathbf{p}_{t+1}^c\mathbf{p}_{t+1}^{c\,T} + c_\mu\mathbf{C}_{t+1}^\mu$,
  $\sigma_{t+1} \leftarrow \sigma_t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_{t+1}^\sigma\|_2}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|_2} - 1\right)\right)$,
  $t \leftarrow t+1$
end while

Algorithm 2 Regularization

function REGULARIZE($\mathbf{C}$, τ) ▷ $\mathbf{C}$ is a covariance matrix
  $\tilde{\mathbf{C}} \leftarrow \text{diag}\mathbf{C}^{-1/2}\mathbf{C}\text{diag}\mathbf{C}^{-1/2}$,
  $\mathbf{P} \leftarrow \tilde{\mathbf{C}}^{-1}$
  $\tilde{\mathbf{P}} \leftarrow \text{diag}\mathbf{P}^{-1/2}\mathbf{P}\text{diag}\mathbf{P}^{-1/2}$,
  $\mathbf{W}_{ij} \leftarrow 1$ if $|\tilde{\mathbf{P}}_{ij}| < \tau$ else 0
  $\mathbf{P}_{\text{reg}} \leftarrow \text{argmin}_{\mathbf{\Theta} \epsilon \mathcal{S}_{++}^n} \text{tr}(\tilde{\mathbf{C}}\mathbf{\Theta}) - \log\det\mathbf{\Theta} + \sum_{i,j=1}^{n} \mathbf{W}_{ij}|\theta_{ij}|$ ▷ Initialized at $\mathbf{P}$
  $\tilde{\mathbf{C}}_{\text{reg}} \leftarrow \mathbf{P}_{\text{reg}}^{-1}$
  return $\text{diag}\mathbf{C}^{1/2}\tilde{\mathbf{C}}_{\text{reg}}\text{diag}\mathbf{C}^{1/2}$
end function

Name	Definition	$\mathbf{x}_0$
Sphere	$f_{\text{sphere}}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$3 \cdot \mathbf{1}$
Ellipsoid	$f_{\text{elli}}(\mathbf{x}) = \sum_{i=1}^{n} 10^{6 \frac{i-1}{n-1}} x_i^2$	$3 \cdot \mathbf{1}$
Cigar	$f_{\text{cig}}(\mathbf{x}) = x_1^2 + \sum_{i=2}^{n} 10^{6 \frac{i-1}{n-1}} x_i^2$	$3 \cdot \mathbf{1}$
Tablet	$f_{\text{tab}}(\mathbf{x}) = 10^6 x_1^2 + \sum_{i=2}^{n} x_i^2$	$3 \cdot \mathbf{1}$
Twoaxes	$f_{\text{twoax}}(\mathbf{x}) = \sum_{i=1}^{\lfloor n/2 \rfloor} 10^6 x_i^2 + \sum_{i=\lfloor n/2 \rfloor + 1}^{n} x_i^2$	$3 \cdot \mathbf{1}$
Subspace Rotated Ellipsoid	$f_{\text{ellisub}}(\mathbf{x}) = \sum_{i=2}^{n-1} 10^{6 \frac{i-1}{n-1}} x_i^2 + \begin{pmatrix} x_1 & x_n \end{pmatrix} \mathbf{R}^T \begin{pmatrix} 1 & 0 \\ 0 & 10^6 \end{pmatrix} \mathbf{R} \begin{pmatrix} x_1 \\ x_n \end{pmatrix}$	$3 \cdot \mathbf{1}$
Rosenbrock	$f_{\text{rosen}}(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$	$\mathbf{0}$
2-Blocks Ellipsoid	$f_{\text{2-blocks elli}}(\mathbf{x}) = f_{\text{elli}}(\mathbf{B}\mathbf{x})$	$3 \cdot \mathbf{1}$
2-Blocks Cigar	$f_{\text{2-blocks cig}}(\mathbf{x}) = f_{\text{cig}}(\mathbf{B}\mathbf{x})$	$3 \cdot \mathbf{1}$
2-Blocks Tablet	$f_{\text{2-blocks tab}}(\mathbf{x}) = f_{\text{tab}}(\mathbf{B}\mathbf{x})$	$3 \cdot \mathbf{1}$
Permuted 2-Block Rotated Ellipsoid	$f_{\text{perm ellisub}}(\mathbf{x}) = f_{\text{elli}}(\mathbf{P}_2 \mathbf{B} \mathbf{P}_1 \mathbf{x})$	$3 \cdot \mathbf{1}$
Rotated Ellipsoid	$f_{\text{ellirot}}(\mathbf{x}) = f_{\text{elli}}(\mathbf{Q}\mathbf{x})$	$3 \cdot \mathbf{1}$
k-Rotated Quadratic	$f_{k\text{-rot}}(\mathbf{x}) = \sum_{i=1}^{n-k} x_i^2 + \begin{pmatrix} x_{n-k+1} & \cdots & x_n \end{pmatrix} \mathbf{R}_k^T \begin{pmatrix} \mathbf{I}_{k-1} & \mathbf{0} \\ \mathbf{0} & 10^6 \end{pmatrix} \mathbf{R}_k \begin{pmatrix} x_{n-k+1} \\ \vdots \\ x_n \end{pmatrix}$	$3 \cdot \mathbf{1}$

Table 3.1: Benchmark functions. The matrix $\mathbf{R}$ ($\mathbf{R}_k$) is a random 2×2 ($k \times k$) rotation matrix drawn from the Haar distribution in SO(2) (SO(k) respectively). The block diagonal matrix $\mathbf{B}$ has the form $\mathbf{B} = \left(\begin{smallmatrix} \mathbf{B}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 \end{smallmatrix} \right)$, where $\mathbf{B}_1$ and $\mathbf{B}_2$ are random rotation matrices of size $\frac{n}{2} \times \frac{n}{2}$ and $\mathbf{Q}$ is a random rotation matrix of size $n \times n$, all drawn from the Haar distribution, while $\mathbf{P}_1, \mathbf{P}_2$ are random permutation matrices.

population additionally with negative recombination weights.

The second test case is the non-convex Rosenbrock function $f_{\text{rosen}}(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$, for which the Hessian matrix is (globally) tridiagonal. Figure 3.3 presents the speed-up obtained for different values of the threshold parameter τ and the scaling with dimension. In dimension $n = 5$ the convergence speed is almost the same with the speed of CMA-ES, while in dimension $n = 80$, the method becomes more than 3 times faster. The conditional dependency graphs learned by the proposed approach for 2 different values of τ are shown in Figure 3.4.

Furthermore, we illustrate the learned conditional dependency pattern for a test function where the number of non zero entries of the Hessian is quadratic with n. In particular, we define $f_{\text{perm ellisub}}(\mathbf{x}) = f_{\text{elli}}(\mathbf{P}_2 \mathbf{B} \mathbf{P}_1 \mathbf{x})$, where $\mathbf{P}_1, \mathbf{P}_2$ are random permutation matrices and $\mathbf{B}$ a 2-block diagonal rotation matrix, see also Table 3.1. Figure 3.5 presents the graph that corresponds to the true Hessian sparsity pattern and the final conditional dependency graph resulting from gl-CMA-ES.

The next example is the function $f_{k\text{-rot}}$, defined in Table 3.1, which results from an ill-conditioned separable function after performing a (random) rotation in a $k-$dimensional subspace of $\mathbb{R}^n$. This forms a group of k strongly dependent search coordinates (with high probability) and the Hessian's sparsity decreases with increasing k. Figure 3.6 illustrates the convergence speed for different threshold values and for varying values of k. Threshold values between 0.3 and 0.5 reveal similar and close to optimal performance.

Finally, the performance scaling on the rest of the benchmark functions of Table 3.1 is shown in Figure 3.7 for selected threshold parameter values, chosen after preliminary experimentation in a way that the estimated preci-

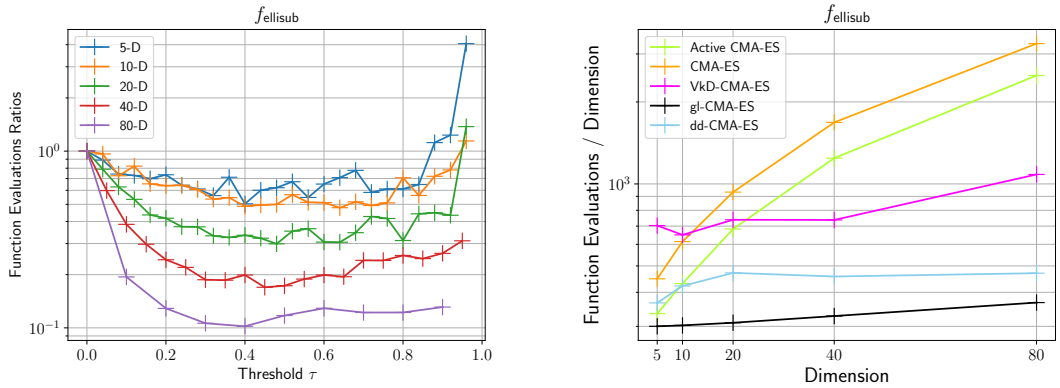

Figure 3.2: Left: Ratios of number of function evaluations performed in single runs to reach a precision 10^{-10} close to the global optimal f value of the proposed approach over CMA-ES depending on the regularization threshold τ. One run has been performed per each value of τ (hence the graphs appear as noisy since there is no averaging of the shown ratios). Right: Scaling with dimension of the average number of function evaluations to reach a 10^{-10} precision. The average is taken over 10 independent runs of each algorithm. The chosen threshold value is $\tau = 0.4$.

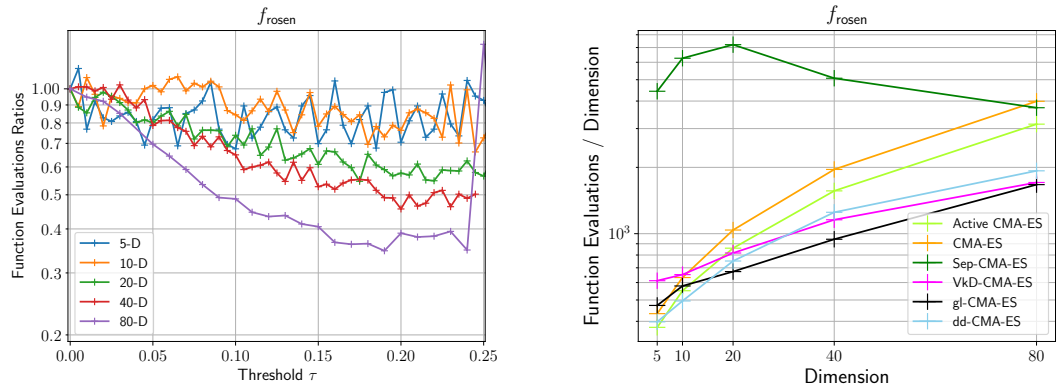

Figure 3.3: Left: Ratios of number of function calls performed in single runs to reach a precision of 10^{-10} of the proposed approach over CMA-ES (without averaging). Right: Performance scaling for $\tau = 0.24$ with averaging over 10 independent runs.

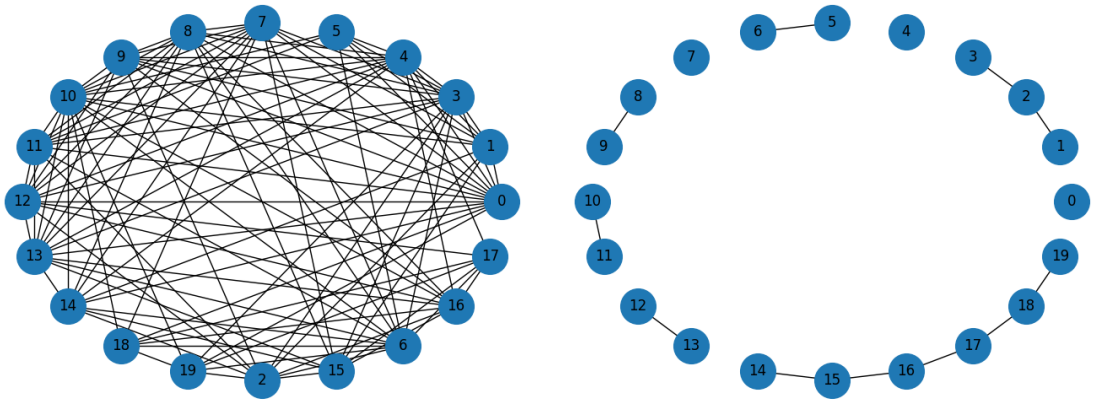

Figure 3.4: Conditional dependency graphs in the last iteration of a single run for thresholds $\tau = 0.05$ (left) and $\tau = 0.24$ (right) on the 20-dimensional Rosenbrock function. Edges depict non zero off-diagonal entries of the precision matrix.

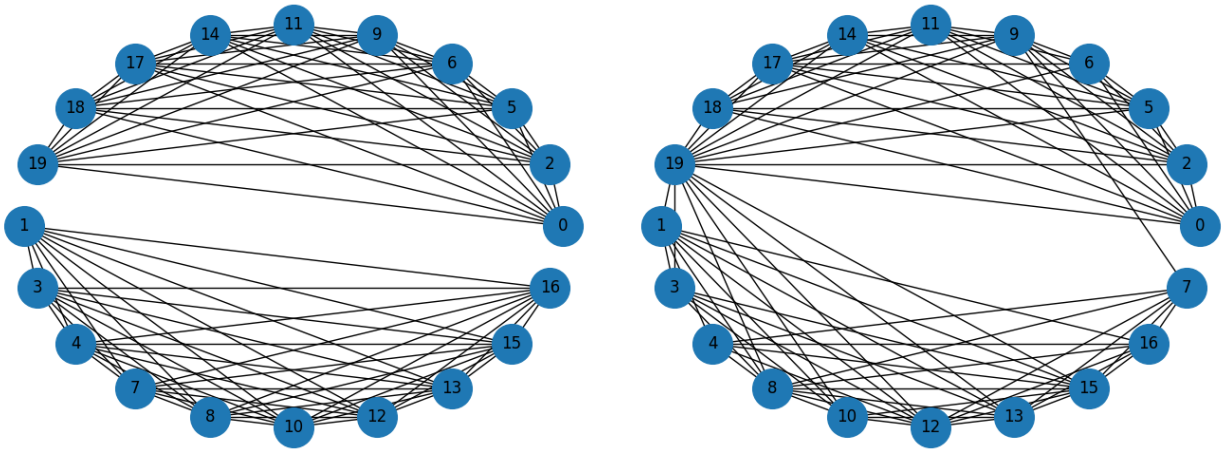

Figure 3.5: Adjacency graph of the true Hessian matrix (left) and conditional dependency graph in the last iteration of a single run of gl-CMA-ES with $\tau = 0.1$ on $f_{\text{perm ellisub}}$.

sion's sparsity pattern is not less rich than the Hessian's true pattern. Separable problems allow a choice of a large value for τ and we obtain a behaviour very similar to Separable CMA-ES. Also, the scaling of the method on sparse non-separable problems such as the $f_{\text{2-blocks tablet}}$ and $f_{\text{2-blocks elli}}$ functions is advantageous over all other methods, with the exception of dd-CMA-ES, which shows better scaling on the latter function, due to less conservative learning rate values compared to gl-CMA-ES. For both functions, in dimension $n = 6$, gl-CMA-ES and CMA-ES differ by a factor smaller than 1.3 and in dimension $n = 80$, gl-CMA-ES is more than twice as fast as CMA-ES. One exception is the $f_{\text{2-blocks cigar}}$ function, where all other methods outperform gl-CMA-ES, for dimensions $n \geq 10$. This is also the only case of worse performance compared to CMA-ES indicating that the choice of τ is too large. On fully dense non-separable problems such as the rotated Ellipsoid function f_{ellirot}, the value $\tau = 0$ reduces to the default setting of CMA-ES and the performance is identical.

3.3 Hard Thresholding

A simpler and straightforward approach instead of solving the Graphical Lasso problem is to apply hard thresholding regularization. This method has been studied extensively in the case of sparse covariance estimation, see for example [16]. In the previous context of estimating a search model for partially separable optimization, it can be applied to the partial correlation matrix. This approach gains attention because of its simplicity, while it is also invariant to permutations and prevents symmetry. The limitation is, though, that positive definiteness can be easily lost after thresholding without any restriction for the threshold value.

Let us denote $\mathcal{T}_\tau$ the hard-thresholding operator, that is for a matrix $\mathbf{M}$:

$$\mathcal{T}_\tau(\mathbf{M})_{ij} = \mathbf{M}_{ij} \mathbb{1}_{|\mathbf{M}_{ij}| \geq \tau}. \tag{3.4}$$

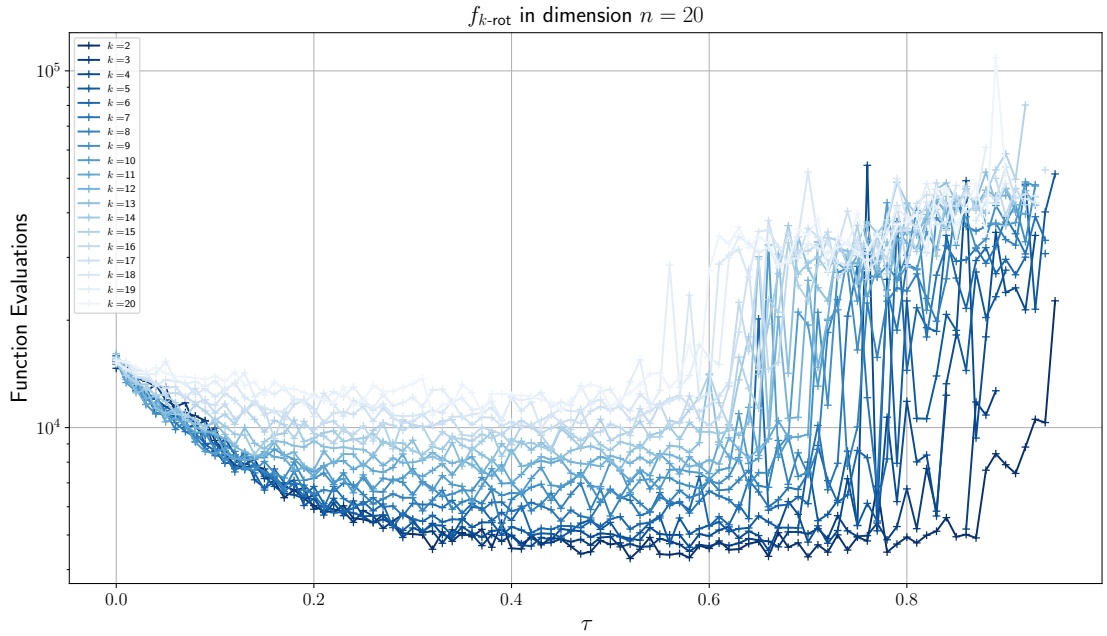

Figure 3.6: Number of function evaluations performed by single runs of gl-CMA-ES to reach the global optimum of $f_{k\text{-rot}}$, for different values of k versus the threshold τ (each value has been tested once and the numbers of function evaluations are not averaged, hence the graphs appear as noisy). The sparsity of the objective function's Hessian is determined by the block size k. Missing values of the number of function evaluations (typically for large threshold values, which lead to an axis-parallel search model) correspond to single runs where gl-CMA-ES does not reach the optimum within a precision of 10^{-10}. The maximal gain in convergence speed is observed when the sparsity is maximal, i.e. for $k = 2$.

Let in addition $||\mathbf{M}||$ denote the operator norm

$$||\mathbf{M}|| = \sup\{||\mathbf{M}\mathbf{x}||_2 : ||\mathbf{x}||_2 = 1\}, \tag{3.5}$$

also given by $||\mathbf{M}|| = \max_{1\leq j\leq n} |\lambda_j(\mathbf{M})|$ if $\mathbf{M} \in \mathcal{S}^n$, $\lambda_j(\mathbf{M})$ denoting the eigenvalues of $\mathbf{M}$ indexed by j. It is straightforward that if $\mathbf{M} \in \mathcal{S}^n_{++}$, a sufficient threshold condition which guarantees positive definiteness [3] of the thresholded matrix $\mathcal{T}_\tau(\mathbf{M})$ is [16]:

$$||\mathcal{T}_\tau(\mathbf{M}) - \mathbf{M}|| < \min_{1\leq j\leq n} |\lambda_j(\mathbf{M})|. \tag{3.6}$$

Algorithm 3 describes a process very similar to the method of the previous section, where simply instead of applying Lasso regularization to the sampling distribution, we apply hard-thresholding to the partial correlation matrix, whose entries serve to measure the degree of conditional dependence of the corresponding search coordinates.

As illustrated in Figure 3.8, the method achieves to resolve even fully dense, ill-conditioned non-separable problems, such as f_{ellirot}. The gain in convergence speed, though, is almost negligible in sparse problems (e.g. f_{elli},

[3] Note that for a given matrix $\mathbf{M} \in \mathcal{S}$, the mapping $s \mapsto ||\mathcal{T}_s(\mathbf{M}) - \mathbf{M}||$ is monotone (and piecewise constant), thus $\sup\{s \geq 0 : ||\mathcal{T}_s(\mathbf{M}) - \mathbf{M}|| < \min_{1\leq j\leq n} |\lambda_j(\mathbf{M})|\}$ can be easily found e.g. via binary search.

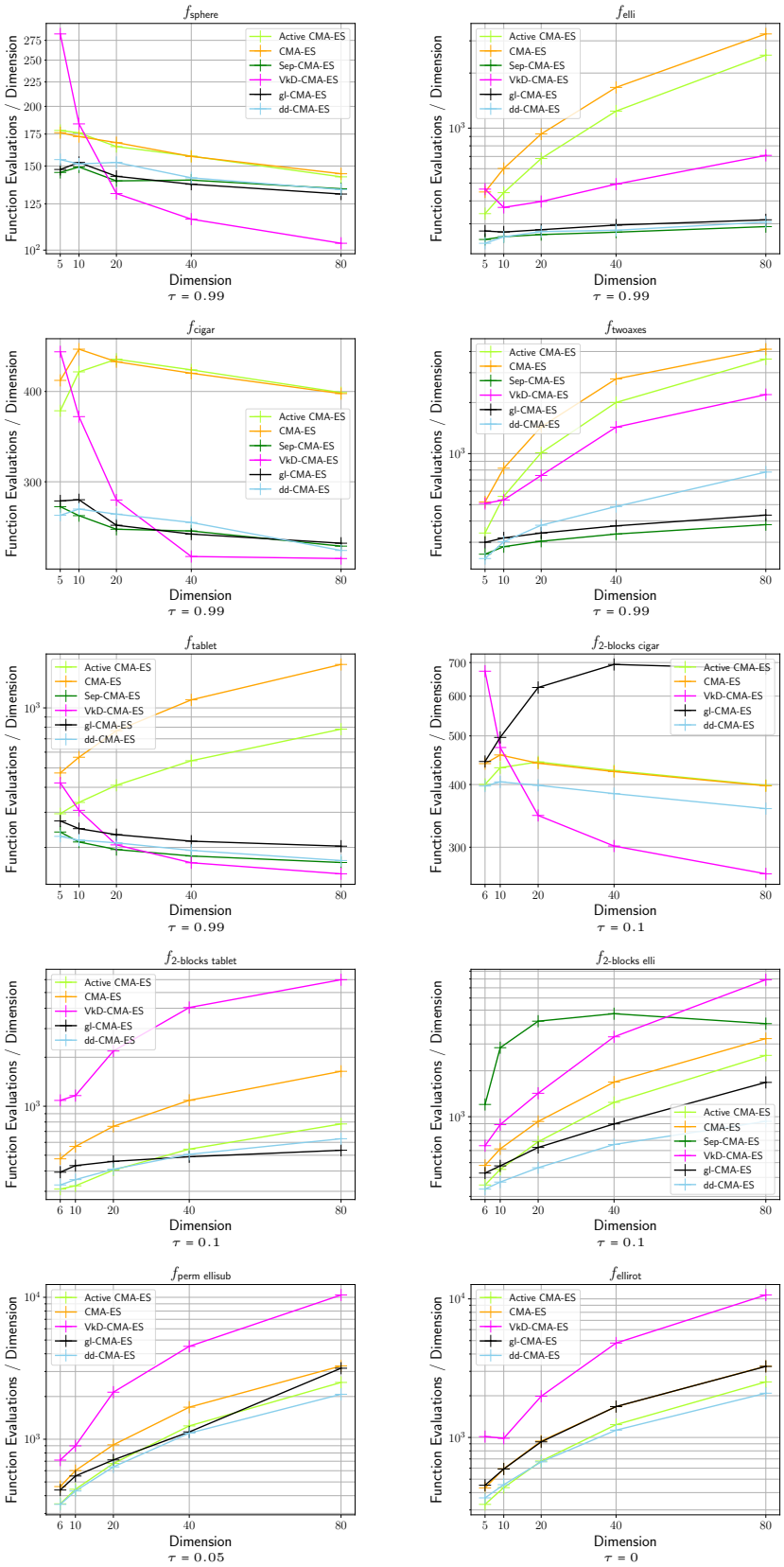

Figure 3.7: Scaling on benchmark functions for selected thresholds. The performance measure is the number of function evaluations to reach a target precision 10^{-10} close to the global optimal f value. Average is taken over 10 independent runs of each method.

f_{ellisub}—see Table 3.1 for the function definitions). The positivity condition (3.6) is too restrictive for the determination of the threshold value and as a result the covariance matrix learning rate is not significantly increased, in comparison to the default values without thresholding.

Algorithm 3 CMA-ES with thresholding

Set parameters: $\lambda = 4 + \lfloor 3 \ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+\frac{1}{2})-\ln i}{\sum_{j=1}^{\mu} \ln(\mu+\frac{1}{2})-\ln j}$ for $i = 1 \dots \mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\mu_w+2}{n+\mu_w+3}$, $d_\sigma = 1 + 2\max\{0, \sqrt{\frac{\mu_w-1}{n+1}} - 1\} + c_\sigma$, $c_c = \frac{4+\mu_w/n}{n+4+2\mu_w/n}$,
Initialize: $\mathbf{p}_t^c \leftarrow \mathbf{0}$, $\mathbf{p}_t^\sigma \leftarrow \mathbf{0}$, $\mathbf{C}_t \leftarrow \mathbf{I}$, $t \leftarrow 0$,
while termination criteria not met **do**
  $\mathbf{P}_t \leftarrow \mathbf{C}_t^{-1}$,
  $\tilde{\mathbf{P}}_t \leftarrow \text{diag}(\mathbf{P}_t)^{-1/2} \mathbf{P}_t \text{diag}(\mathbf{P}_t)^{-1/2}$,
  $\tilde{\mathbf{P}}^{\text{reg}} \leftarrow \text{THRESHOLD}(\tilde{\mathbf{P}}_t)$,
  $\mathbf{C}_{t+1}^{\text{reg}} \leftarrow \text{diag}(\mathbf{P}_t)^{-1/2} \tilde{\mathbf{P}}^{\text{reg}^{-1}} \text{diag}(\mathbf{P}_t)^{-1/2}, n_z \leftarrow \#|\tilde{\mathbf{P}}^{\text{reg}}| > 0$,
  $c_1 \leftarrow \frac{2}{(n_z/n+1.3)(n+1.3)+\mu_w}, c_\mu \leftarrow \min\{1-c_1, 2\frac{\mu_w+1/\mu_w-1.75}{(n_z/n+2)(n+2)+\mu_w}\}$,

  for $k \leftarrow 1, \dots, \lambda$ **do**
    $\mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_{t+1}^{\text{reg}})$
    $\mathbf{x}_k \leftarrow \mathbf{m}_t + \sigma_t \mathbf{z}_k$
    $f_k \leftarrow f(\mathbf{x}_k)$
  end for
  $\mathbf{m}_{t+1} \leftarrow \sum_{k=1}^{\mu} w_k \mathbf{x}_{k:\mu}$
  $\mathbf{p}_{t+1}^\sigma \leftarrow (1-c_\sigma)\, \mathbf{p}_t^\sigma + \sqrt{c_\sigma(2-c_\sigma)\mu_w}\, \mathbf{C}_{t+1}^{\text{reg}\,-\frac{1}{2}} \frac{\mathbf{m}_{t+1}-\mathbf{m}_t}{\sigma_t}$,
  $h_\sigma \leftarrow \mathbb{1}_{\|\mathbf{p}_{t+1}^\sigma\|_2 < (1.4+\frac{2}{n+1})\sqrt{1-(1-c_\sigma)^{2(t+1)}}\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|_2}$,
  $\delta(h_\sigma) \leftarrow (1-h_\sigma) c_c (2-c_c)$,
  $\mathbf{p}_{t+1}^c \leftarrow (1-c_c)\, \mathbf{p}_t^c + h_\sigma \sqrt{c_c(2-c_c)\mu_w} \frac{\mathbf{m}_{t+1}-\mathbf{m}_t}{\sigma_t}$,
  $\mathbf{C}_{t+1}^\mu \leftarrow \sum_{k=1}^{\mu} w_k \mathbf{z}_{k:\mu} \mathbf{z}_{k:\mu}^T$,
  $\mathbf{C}_{t+1} \leftarrow (1 + c_1\delta(h_\sigma) - c_1 - c_\mu)\mathbf{C}_t + c_1 \mathbf{p}_{t+1}^c \mathbf{p}_{t+1}^{c\,T} + c_\mu \mathbf{C}_{t+1}^\mu$,
  $\sigma_{t+1} \leftarrow \sigma_t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_{t+1}^\sigma\|_2}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|_2} - 1\right)\right)$,
  $t \leftarrow t+1$
end while

Algorithm 4 Hard thresholding

function THRESHOLD($\mathbf{M}$)
  $\tau \leftarrow \sup\{s \geq 0 : \|\mathcal{T}_s(\mathbf{M}) - \mathbf{M}\| + 10^{-9} < \min_{1 \leq j \leq n} |\lambda_j(\mathbf{M})|\}$
  $\mathbf{M}_{ij} \leftarrow \mathbf{M}_{ij} \mathbb{1}_{|\mathbf{M}_{ij}| > \tau}$
  return $\mathbf{M}$
end function

3.4 Sparse precision via single-link updates

Martin et al. propose in [62] a greedy algorithm for the estimation of a Gaussian Markov Random Field (GMRF) with a sparse precision matrix. The method was originally oriented to address the problem of estimating a GMRF which

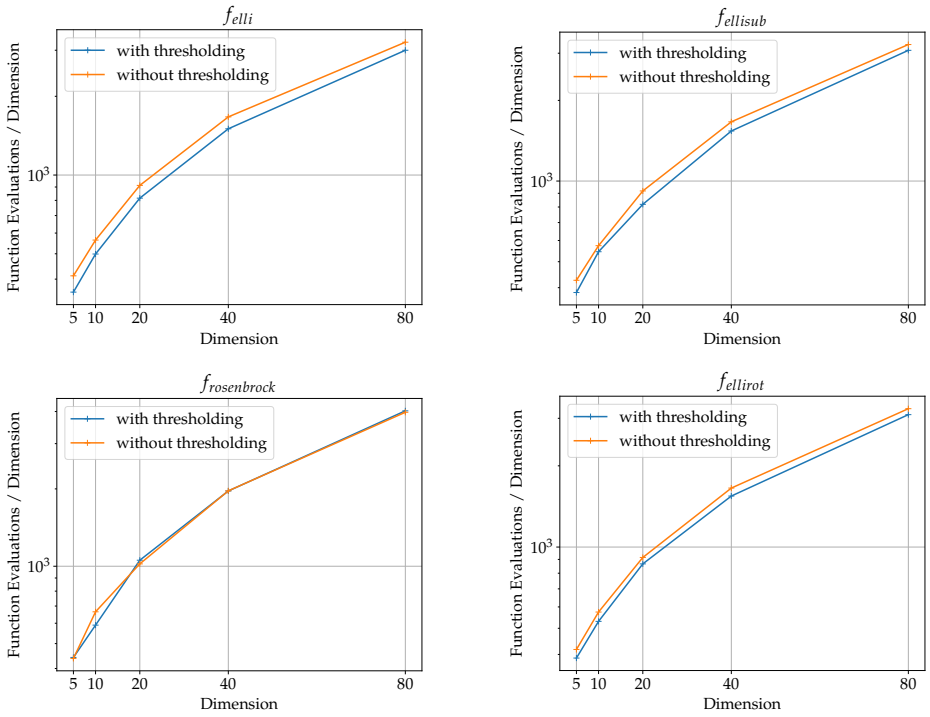

Figure 3.8: Scaling graphs of number of f-evaluations divided by dimension to reach the global optimum up to a precision of 10^{-10} for selected test functions. Average is taken over 5 runs.

satisfies constraints imposed from a given dependency graph structure, such that it is compatible with the Gaussian Belief Propagation algorithm [17]. Elementary steps of this method can be applied in our context of accelerating the CMA-ES adaptation, essentially by considering single-link updates as described in the following.

Similarly to the section 3.2, we consider the problem

$$\underset{\mathbf{X}\in\mathcal{S}_{++}^{n}}{\text{maximize}} \log\det\mathbf{X} - \mathrm{Tr}(\mathbf{XC}), \tag{3.7}$$

$\mathbf{X}$ representing the precision matrix to be estimated, which is equivalent to minimizing the Kullback-Leibler divergence:

$$\underset{\mathbf{X}\in\mathcal{S}_{++}^{n}}{\text{minimize}} D_{KL}(\mathcal{N}(\mathbf{0},\mathbf{C})\|\mathcal{N}(\mathbf{0},\mathbf{X}^{-1})) \tag{3.8}$$

and trivially has the solution $\mathbf{X} = \mathbf{C}^{-1}$ if no constraints are imposed. If a reference distribution density $\mathcal{P}^{\text{old}}$ is given along with a set of empirical marginals $\tilde{p}_{ij}$ and we perform a single link correction represented by ψ_{ij} to produce the updated density $\mathcal{P}^{\text{new}}$, i.e.

$$\mathcal{P}^{\text{new}}(\mathbf{x}) = \mathcal{P}^{\text{old}}(\mathbf{x})\psi_{ij}(x_i, x_j), \tag{3.9}$$

then the index pair (k, l) (with $k \neq l$) that maximizes

$$\Delta\mathcal{L}_{ij} = D_{KL}(\tilde{p}_{ij} \| p_{ij}^{\text{old}}) = \int_{\mathbb{R}^2} \tilde{p}_{ij}(u, v) \log \frac{\tilde{p}_{ij}(u, v)}{p_{ij}^{\text{old}}(u, v)} du dv, \tag{3.10}$$

corresponds to the optimal one-link correction (as the pair which yields the most divergent marginal p_{ij}^{old} from $\tilde{p}_{ij}$), which would be given by

$$\psi_{kl}(x_k, x_l) = \frac{\tilde{p}_{kl}(x_k, x_l)}{p_{kl}^{\text{old}}(x_k, x_l)}. \tag{3.11}$$

In this case, if the empirical distribution is specified by a precision matrix $\tilde{\mathbf{P}} = \tilde{\mathbf{C}}^{-1}$ and the reference distribution by $\mathbf{P}^{\text{old}} = \mathbf{C}^{\text{old}^{-1}}$, the updated precision reads [62]:

$$\mathbf{P}^{\text{new}} = \mathbf{P}^{\text{old}} + [\tilde{\mathbf{C}}_{\{kl\}}^{-1}] - [\mathbf{C}_{\{kl\}}^{\text{old}}{}^{-1}] \tag{3.12}$$

where as in [62] we denote $\mathbf{C}_{\{kl\}}^{\text{old}}$ ($\tilde{\mathbf{C}}_{\{kl\}}$) the restricted 2×2 covariance matrix of the marginal distribution of the pair (x_k, x_l), obtained after extracting the corresponding entries from the reference covariance matrix $\mathbf{C}^{\text{old}}$ (empirical covariance matrix $\tilde{\mathbf{C}}$ respectively). The $n \times n$ matrix $[\tilde{\mathbf{C}}_{\{kl\}}^{-1}]$ ($\mathbf{C}_{\{kl\}}^{\text{old}}{}^{-1}$) is obtained by completing $\tilde{\mathbf{C}}_{\{kl\}}^{-1}$ ($\mathbf{C}_{\{kl\}}^{\text{old}}{}^{-1}$ respectively) with zeros.

The log-likelihood variation $\Delta\mathcal{L}_{kl}$ reads:

$$\Delta\mathcal{L}_{kl} = \frac{\mathbf{C}_{kk}^{\text{old}}\tilde{\mathbf{C}}_{ll} + \mathbf{C}_{ll}^{\text{old}}\tilde{\mathbf{C}}_{kk} - 2\mathbf{C}_{kl}^{\text{old}}\tilde{\mathbf{C}}_{kl}}{\det(\mathbf{C}_{kl}^{\text{old}})} - \log \frac{\det(\tilde{\mathbf{C}}_{kl})}{\det(\mathbf{C}_{kl}^{\text{old}})} - 2 \tag{3.13}$$

and the single-link update of the covariance matrix, using (3.12) and the Sherman-Morrison-Woodbury identity translates as:

$$\mathbf{C}^{\text{new}} = \mathbf{C}^{\text{old}} - \mathbf{C}^{\text{old}}[\mathbf{C}_{\{kl\}}^{\text{old}}{}^{-1}](\mathbf{I} - [\tilde{\mathbf{C}}_{\{kl\}}^{-1}][\mathbf{C}_{\{kl\}}^{\text{old}}{}^{-1}])\mathbf{C}^{\text{old}}. \tag{3.14}$$

Within the CMA-ES context, we can integrate single-link corrections instead of fully update the covariance matrix, by considering the set of empirical marginals as those that result from the search distribution after selection and recombination. For simplicity, in the folllowing we disregard the rank-one update and we consider recombination with positive weights. In this case, equation (2.3) can be rewritten as

$$\mathbf{C}_{t+1} = \mathbf{C}_t + c_\mu \left(\sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda} \mathbf{y}_{i:\lambda}{}^T - \mathbf{C}_t \right) \tag{3.15}$$

since the weights satisfy $\sum_{i=1}^{\mu} w_i = 1$ by default. The rank-μ update term can be compactly written as $\mathbf{V}\mathbf{V}^T$, with $\mathbf{V} = \left[\sqrt{w_1}\mathbf{y}_{1:\lambda} \ \dots \ \sqrt{w_\mu}\mathbf{y}_{\mu:\lambda}\right]$ and the update equation reads

$$\mathbf{C}_{t+1} = \mathbf{C}_t + c_\mu \left(\mathbf{V}\mathbf{V}^T - \mathbf{C}_t \right). \tag{3.16}$$

Using the notation above, we consider the reference $\mathbf{C}^{\text{old}} = \mathbf{C}_t$ and the empirical precision $\tilde{\mathbf{P}} = (\mathbf{V}\mathbf{V}^T)^{-1}$ and covariance $\tilde{\mathbf{C}} = \mathbf{V}\mathbf{V}^T$ to produce $\mathbf{C}^{\text{new}}$ via (3.14). Instead of performing steps in the natural gradient direction $\mathbf{V}\mathbf{V}^T - \mathbf{C}_t$, the modified steps are

$$\mathbf{C}_{t+1} = \mathbf{C}_t + c_\mu \left(\mathbf{C}^{\text{new}} - \mathbf{C}_t\right). \tag{3.17}$$

Note that in order for (3.9) to be meaningful and for $\tilde{\mathbf{C}}$ to be a.s. non-degenerate, the population size must be chosen such that $\lambda \geq \mu \geq n$. Furthermore, in accordance to the learning rate setting of the first section, we choose $c_\mu = \Theta(1)$. The method is summarized in Algorithm 5. Figure 3.9 shows the behaviour of the algorithm on selected sparse problems.

One limitation is that when a new link addition according to 3.12 is performed, the neighbouring links of the newly joint nodes are detuned. This can have a significant impact on the performance, if the detuned links correspond to pairs of coordinates strongly dependent. A remedy to this effect has been proposed in [62], by performing link corrections via row-column updates, also proposed in [14, 31]. These approaches are not further investigated here and are left as future work. Figure 3.10 shows the scaling of sl-CMA-ES on f_{ellisub} (see also table 3.1), where its performance is promising, as well as the method's behaviour on f_{ellirot} where it is unsuccessful.

Algorithm 5 sl-CMA-ES

Set parameters: $\lambda = 2n$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+\frac{1}{2})-\ln i}{\sum_{j=1}^{\mu} \ln(\mu+\frac{1}{2})-\ln j}$ for $i = 1 \dots \mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\mu_w+2}{n+\mu_w+3}$, $d_\sigma = 1 + 2\max\{0, \sqrt{\frac{\mu_w-1}{n+1}} - 1\} + c_\sigma$, $c_\mu \leftarrow \Theta(1)$,
Initialize: $\mathbf{p}_t^\sigma \leftarrow \mathbf{0}$, $\mathbf{C}_t \leftarrow \mathbf{I}$, $t \leftarrow 0$,
while termination criteria not met **do**
  for $k \leftarrow 1, \dots, \lambda$ **do**
    $\mathbf{z}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_t)$
    $\mathbf{x}_k \leftarrow \mathbf{m}_t + \sigma_t \mathbf{z}_k$
    $f_k \leftarrow f(\mathbf{x}_k)$
  end for
  $\mathbf{m}_{t+1} \leftarrow \sum_{k=1}^{\mu} w_k \mathbf{x}_{k:\mu}$
  $\mathbf{p}_{t+1}^\sigma \leftarrow (1-c_\sigma)\,\mathbf{p}_t^\sigma + \sqrt{c_\sigma(2-c_\sigma)\mu_w}\,\mathbf{C}_t^{-\frac{1}{2}} \frac{\mathbf{m}_{t+1}-\mathbf{m}_t}{\sigma_t}$,
  $h_\sigma \leftarrow \mathbb{1}_{\|\mathbf{p}_{t+1}^\sigma\|_2 < (1.4+\frac{2}{n+1})\sqrt{1-(1-c_\sigma)^{2(t+1)}}\,\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|_2}$,
  $\delta(h_\sigma) \leftarrow (1-h_\sigma)c_c(2-c_c)$,
  $\mathbf{V} = \left[\sqrt{w_1}\mathbf{z}_{1:\lambda}\ \dots\ \sqrt{w_\mu}\mathbf{z}_{\mu:\lambda}\right]$, $\tilde{\mathbf{C}} \leftarrow \mathbf{V}\mathbf{V}^T$,
  $\Delta\mathcal{L}_{ij} \leftarrow \frac{(\mathbf{C}_t)_{ii}\tilde{\mathbf{C}}_{jj}+(\mathbf{C}_t)_{jj}\tilde{\mathbf{C}}_{ii}-2(\mathbf{C}_t)_{ij}\tilde{\mathbf{C}}_{ij}}{\det((\mathbf{C}_t)_{\{i,j\}})} - \log\frac{\det(\tilde{\mathbf{C}}_{\{i,j\}})}{\det((\mathbf{C}_t)_{\{i,j\}})}$-2
  $(k,l) \leftarrow \text{argmax}_{(i,j)\in\{1,\dots,n\}^2, i\neq j}\, \Delta\mathcal{L}_{ij}$,
  $\mathbf{C}_{t+1} \leftarrow \mathbf{C}_t - c_\mu \mathbf{C}_t [(\mathbf{C}_t)_{\{kl\}}{}^{-1}] \left(\mathbf{I} - [\tilde{\mathbf{C}}_{\{kl\}}^{-1}][(\mathbf{C}_t)_{\{kl\}}{}^{-1}]\right) \mathbf{C}_t$,
  $\sigma_{t+1} \leftarrow \sigma_t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_{t+1}^\sigma\|_2}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|_2} - 1\right)\right)$,
  $t \leftarrow t+1$
end while

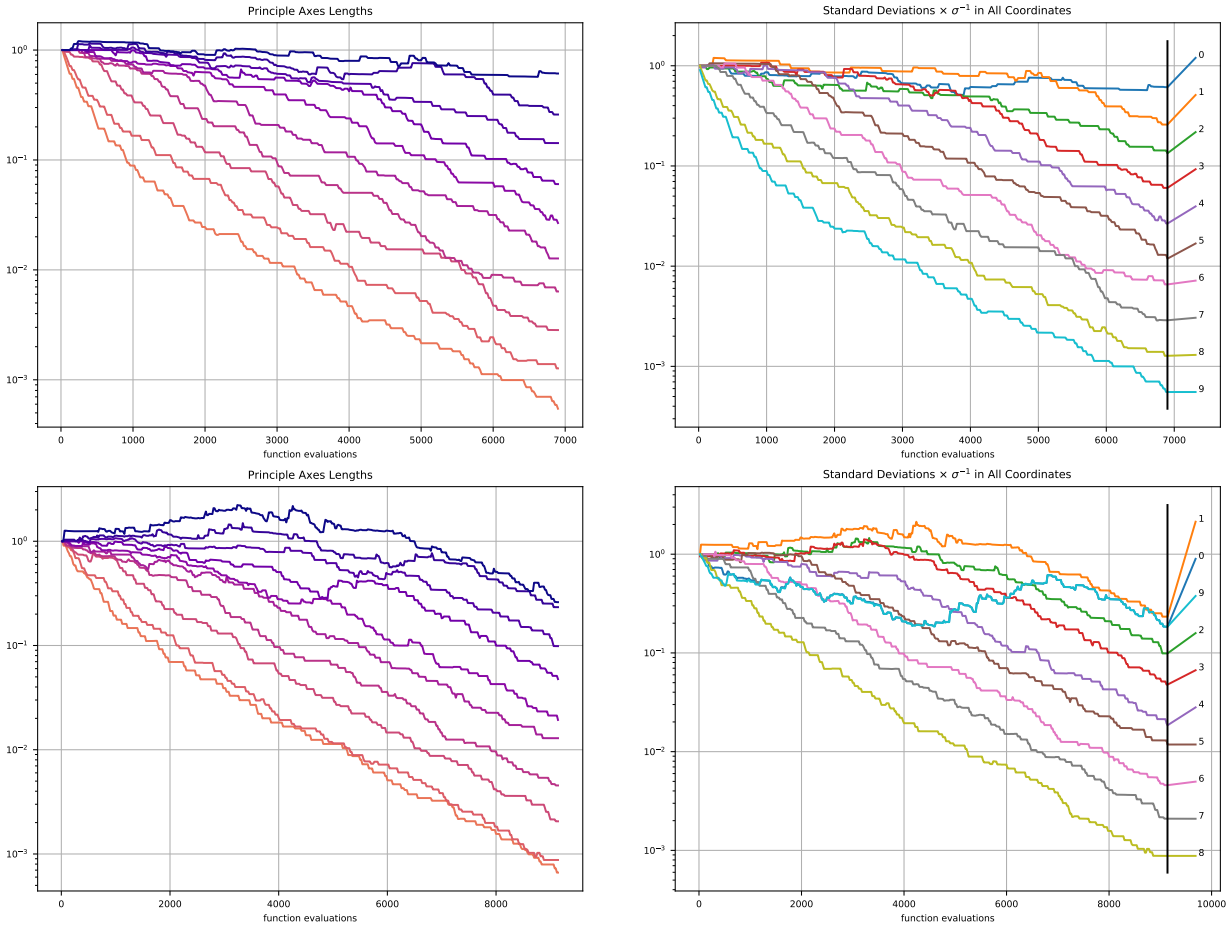

Figure 3.9: Principal axes lengths (left) and standard deviations (right) normalized by the step size for single runs of sl-CMA-ES on f_{elli} (above) and f_{ellisub} (below) in dimension 10 (see table 3.1 for the function definitions). The covariance matrix learning rate was chosen $c_\mu = 0.2$.

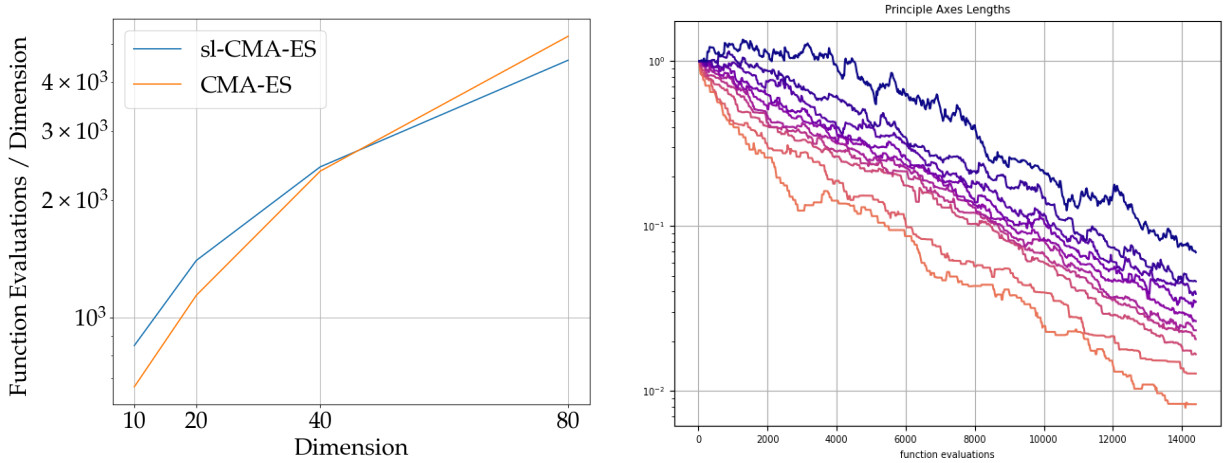

Figure 3.10: Left: Scaling graph of number of f-evaluations of sl-CMA-ES to reach the optimum of f_{ellisub} up to a precision of 10^{-10}, with $c_\mu = 0.2$, compared with CMA-ES. The latter is tested with a population size $\lambda = 2n$ and with positive recombination weights of the covariance matrix update. Average is taken over 5 runs. Right: Principal axes lenths for f_{ellirot}: sl-CMA-ES does not learn the correct distribution, leading to performance deterioration.

3.5 Discussion

In our first method we integrated l_1 regularization within CMA-ES, attempting to increase the adaptation speed of the search distribution. We investigated its behaviour and showed the gain in convergence speed in representative sparse problems and for selected values of the threshold parameter. The setting of the threshold is crucial for the richness of the search model and thus for the performance of gl-CMA-ES, and good choices depend on properties of the function to be optimized, for example separability. As a result, a future step for improving this approach is the inclusion of an adaptive mechanism for this parameter, rather than being predefined and static. Furthermore, we only investigated the performance gain in terms of convergence speed through accelerating the covariance matrix adaptation. No focus has been given on the internal cost of the regularization step using the Graphical Lasso. A possible strategy would be to perform this step in the updated search distribution once every a certain number of iterations while sampling with the same regularized search model in between, in cases where the dimension is large and the computational cost becomes a burden. Also, in order to guarantee the positive definiteness of the covariance matrix, only positive recombination weights are used, as mentioned in the algorithm's description. Therefore, another interesting aspect and future step for improvement is to employ negative recombination weights.

Furthermore, the hard-thresholding technique was tested as an alternative to the Graphical Lasso regularization step. Even though this method does not require any parameter tuning and also solves dense non-separable ill-conditioned functions, it did not provide sufficient gain in comparison to the performance of CMA-ES.

Lastly, imposing sparsity by performing single-link updates might be advantageous in certain sparse problems. Nevertheless, these suboptimal covariance matrix updates may fail and in a black-box scenario, where we don't assume any prior knowledge on the sparsity properties of the function, further steps are required. Therefore, the row-column corrections that we mentioned might be an interesting aspect to investigate, though in principle they might also limit the adaptation speed-up, since the number of learned parameters would depend on the connectivity of the obtained dependency graph.

Chapter 4

Benchmarking

Benchmarking continuous optimization algorithms is essential not only for their performance evaluation but also for detecting defects and improving new methods. A platform which automatizes this process is the Comparing Continuous Optimizers [43, 44] (COCO) platform, containing several test suites with single-objective, bi-objective, noisy or large-scale problems.

This chapter is divided in two parts: the first describes the methodology of assessing the algorithms' performance adopted by COCO, presents related work to large-scale optimization benchmarking and introduces the `bbob-largescale` suite of COCO in its finalised form, a novel suite compatible with the widely-used `bbob` suite [38]. The definition of this suite as an extension of `bbob`, attempts to maintain the "important" characteristics of each test function, while maintaining a low computational cost of evaluating the test function values. In order to achieve this, the functional transformations originally used for the definition of the `bbob` test functions, are restricted to have a certain structure, as was initially introduced in [4] and described in the following [1]. The second part of the chapter includes two benchmarking studies [96, 95], with the experimental evaluation and comparison of several solvers (using both the `bbob` and the `bbob-largescale` suite).

[1] The development of the `bbob-largescale` suite was already ongoing at the starting period of this thesis. However, certain modifications were necessary in order to arrive at its finalised form. Although the first part of the chapter gives a coherent description of benchmarking large scale solvers with this suite, including the presentation of the COCO platform's assessment methodology or recalling parts of the `bbob-largescale` suite that were already implemented (such as the fundamental permuted block-diagonal transformations), we attempt to emphasize on contributions that were made during this thesis (adjustments of particular function definitions for scalability assessment, modifications which achieve compatibility of the `bbob` and `bbob-largescale` suite, data collection, experimental and postprocessing user guides).

4.1 Benchmarking large-scale optimizers[2]

4.1.1 Introduction

Benchmarking is an important task in optimization that every algorithm designer has to do to validate a new algorithm. It can also assist the designer by pointing out weaknesses that have been overlooked in the first conception phase of the algorithm. The choice of the test functions is crucial as performance is often aggregated over sets of functions and a bias towards certain properties can lead to a misrepresentation of the "real" performance of an algorithm.

Optimization problems with more than one hundred variables are common in many domains. We therefore naturally need benchmarking suites to test algorithms in these dimensions and to investigate their scalability.

This section introduces a new benchmarking test suite with the following objectives.

- We extend the widely used Black-Box Optimization Benchmarking suite [38], `bbob`, to larger dimensions. The `bbob` suite is part of the Comparing Continuous Optimizers benchmarking platform [43, 44], COCO , a general tool for benchmarking continuous solvers. The suite has been widely used for the performance comparison of various types of solvers (deterministic, stochastic, evolutionary, gradient-free, gradient-based, etc.), see e.g. [39, 13, 72, 89, 96].

- We allow to investigate the scaling of algorithms up to dimension 640 in a quantitative way, based on the standardized experimental setup of COCO. A unique feature of our proposal is that the presented suite is an extension of the well-established COCO platform with its corresponding advantages: it offers a thought-out, standardized experimental setup, facilitates the automated processing of results (see the introduction of section 4.1.3), uses the number of function evaluations for the quantitative assessment of the performance and of the scaling with dimension on the highest possible measurement scale (see "Runtime and Target Values" in section 4.1.3), and allows to easily collect and compare algorithm performance data from different sources (see the introduction of section 4.1.5 and "Postprocessing" in section 4.1.6). In addition, the new suite naturally extends the dimensionality of the original `bbob` problems where overlapping dimensions allow to verify that the two suites are compatible (see the introduction of section 4.1.4).

The `bbob-largescale` test functions are using in their definition the so-called permuted block-diagonal orthogonal transformations, a set of transformations with sparsity properties originally introduced in [4], which maintain a low computational cost of evaluating the function values. We recall the structure of such transformations and discuss in detail the adjustments needed and decisions taken to arrive at the final test suite. These adjustments

[2]This section is based on the article "Benchmarking large-scale continuous optimizers: the `bbob-largescale` testbed, a COCO software guide and beyond" [99] by K. Varelas, O. Ait El Hara, D. Brockhoff, N. Hansen, D. M. Nguyen, T. Tušar and A. Auger, published to the Applied Soft Computing journal.

are necessary to be backwards compatible with the bbob test suite and to avoid artificial biases towards certain algorithms or algorithm settings (like optima too close to the origin because of normalization factors).

Additionally, we illustrate how to use the new test suite in the context of the COCO platform to be able to benchmark a novel algorithm. In Sections 4.1.6 and 4.1.7 we provide a software user guide, show the plots that are automatically producible with COCO and outline which scientific information we can gather from them.

4.1.2 Related work

In this subsection we introduce the bbob test suite and discuss related work in large-scale benchmarking.

The BBOB test suite

The testbed we will introduce later is based on the Black-Box Optimization Benchmarking test suite (bbob, [38]) of the COCO platform [43, 44], introduced in 2009. The bbob test suite was constructed with the idea to provide

- functions that represent well-known difficulties in continuous optimization, namely non separability, multimodality, ill-conditioning and landscape ruggedness;
- transformations to make functions look less regular, because we do not expect that many real world problems can be expressed in simple and closed mathematical formulas;
- function pairs and groups that allow to test specific properties of an algorithm (for instance, “does the algorithm exploit separability?”);
- a wide range of challenging test problems to reduce the risk of overfitting and to challenge algorithms as much as possible.

In comparison to other well-known test function suites (for example the CUTEr/CUTEst suite [20] [34]), the bbob functions are mostly non-convex and non-smooth. The bbob test suite is structured into five function groups, namely separable functions, functions with low or moderate conditioning, unimodal functions with high conditioning, multimodal functions with adequate global structure, and multimodal functions with weak global structure. Since the notion of separability can be formulated mathematically in various ways, we hereby adopt the following definition: a function $f : \mathbb{R}^n \to \mathbb{R}$ is called separable if it can be expressed as:

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(x_i) \tag{4.1}$$

for some functions $f_i : \mathbb{R} \to \mathbb{R}$, that is, if it is additively decomposable into the sum of univariate functions of single coordinates.

Each bbob function group contains 5 functions except the second one that contains four functions. This balance between the number of functions per group is important to keep in mind when interpreting aggregated performance results.

An additional important aspect of the bbob functions is their scalability: every function has an analytic expression and is defined for an arbitrary dimension. This suggests that the bbob test suite could be used to test "large"-scale algorithms. Yet there is a practical limitation of the original bbob test suite that precludes its usage for dimensions larger than a few hundreds of variables: many of the bbob functions involve matrix multiplications with dense matrices to make them non-separable. More precisely, these bbob functions are constructed in an onion-like fashion as:

$$f(\mathbf{x}) = F_1 \circ F_2 \circ \ldots \circ F_k \circ f_{\text{raw}} \circ T_1 \circ T_2 \circ \ldots T_l(\mathbf{x}) \tag{4.2}$$

where f_{raw} is the underlying raw objective function, for example the ellipsoid function $f_{\text{elli}}(\mathbf{x}) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2$, the F_i are objective space transformations of the form $F_i : \mathbb{R} \to \mathbb{R}$, and the T_i are search space transformations of the form $T_i : \mathbb{R}^n \to \mathbb{R}^n$. Examples of such search space transformations are simple translations and search space *rotations* $T_{\mathbf{R}} : \mathbf{x} \mapsto \mathbf{R}\mathbf{x}$ with $\mathbf{R}$ being an orthogonal matrix in $\mathbb{R}^{n \times n}$.

Orthogonal matrices, that we also refer to as rotation matrices, are at the core of the constructions of many benchmark functions. They allow to have a simple writing of the functions while not favoring a specific representation of the problem (the representation given by the original coordinate system): we can start from a separable function that is typically easy to write and to comprehend and we rotate it to get a non-separable function [82]. This way, we keep the simplicity of the writing of separable functions but take out the separability bias. This construction is scalable. Yet, if a dense orthogonal matrix is used, the matrix vector product calculation is quadratic in the problem dimension and the computation becomes too prohibitive when having, say, more than a few hundred variables and hundreds of problem instances.

For this reason, the idea to replace orthogonal matrices by *sparse orthogonal* matrices has been introduced in [4] to build benchmark functions in large dimensions. Each dense orthogonal matrix is thereby replaced by a permuted block matrix $\mathbf{P}_1\mathbf{B}\mathbf{P}_2$ with only a linear (in the dimension) number of non-zero coefficients where $\mathbf{P}_1$ and $\mathbf{P}_2$ are permutation matrices and $\mathbf{B}$ is a block-diagonal matrix. The reason for using such so-called *permuted orthogonal block-diagonal matrices* in the context of large-scale optimization benchmarking is two-fold: on the one hand, the computation time for the test functions becomes linear in the problem dimension instead of quadratic, resulting in reasonable computation times, on the other hand, we also reckon that real-world problems in large dimensions typically have less than quadratically many degrees of freedom and a test problem construction via sparse orthogonal matrices will automatically keep the number of variable dependencies lower than quadratic.

Large-scale benchmarking

A few test suites for benchmarking numerical optimizers have been around for some time. In the context of large-scale optimization, most notably developed by the "classical" optimization community, are the COPS 3.0 problems [19] and the general CUTEr/CUTEst problems [20] [34].

The Constrained Optimization Problem Set (COPS) 3.0 test suite contains 22 large-scale problems with 398 to 19240 variables, some of which can be used in arbitrary dimension while others are only defined for very specific dimensions. Despite the suite's name, three of the COPS problems are unconstrained. The CUTEr/CUTEst library, on the other hand, contains many more problems (more than 1000), with 378 of them being unconstrained. Of those, 184 problems are available in any dimension and can thus be used to benchmark large-scale optimization algorithms in principle. From these 184 scalable unconstrained problems, finally only 73 of them are not constant, linear, quadratic, or of a sum of squares type.

In the evolutionary computation community, large-scale competitions have been organized at the CEC conference from which three large-scale test suites evolved over time:

- The CEC 2008 suite [90] with 7 functions: shifted Sphere, shifted Schwefel's Problem 2.21, shifted Rosenbrock, shifted Rastrigin, shifted Griewank, shifted Ackley and FastFractal "DoubleDip", tested in three different dimensions.

- The CEC 2010 suite [91] with 20 functions in total and 6 *underlying* functions: Sphere, rotated Ellipsoid, Schwefel's Problem 1.2, Rosenbrock, rotated Rastrigin, and rotated Ackley. These basic functions are combined with no/partial/full rotations to create the 20 functions overall. The competition was setup with the single dimension 1000.

- The CEC 2013 suite [55], based on the CEC 2010 suite, with additional `bbob` transformations, nonuniform subcomponent sizes, imbalance in the contribution of subcomponents and functions with overlapping subcomponents. The competition was setup with the single dimension 1000.

The CEC competitions are setup with a single or small number of different dimensions (although the problems are, in principle, scalable) and the performance assessment is prescribed for a few given budgets and also for three given targets in the CEC 2010 case. This setup does not allow to reliably measure scaling behavior with dimension—one of the most important characteristics a benchmarking experiment for large-scale algorithms should investigate. A different setup was followed in [48], where test functions from the CEC 2010 test suite were used with adjusted dimensions and budgets.

The benchmark suite introduced in [61] consists of a subset of the test functions introduced in the CEC competitions together with additional test functions. In particular, the functions from the CEC 2008 competition without the FastFractal "DoubleDip" function, 5 (shifted) functions, namely the Schwefel's Problem 2.22, the Schwefel's Prob-

lem 1.2, the extended Schaffer function, the Bohachevsky and the Schaffer function, as well as hybrid composition functions built from them formed a testbed of 19 problems in total [45]. In contrast to CEC, the performance was assessed for 5 different dimensions between 50 and 1000, for a given budget and with independent restarts. The performance criterion was the distance between the best achieved and the optimal function value.

Similar to the COPS and CUTEr/CUTEst problems, also for the CEC problems, no effort was spent on investigating whether target difficulties are comparable over problems and dimensions, however, this similarity is necessary to aggregate performances properly over different problems and to investigate the scaling behavior with the problem dimension.

None of the mentioned test suites is furthermore implemented to allow for an *automated benchmarking*, during which the performance data are recorded automatically, to relieve the user from the burden of implementing this tedious task. We address the automated benchmarking issue and the above mentioned shortcomings of the currently available test suites for large-scale (nonlinear or black-box) optimization benchmarking by proposing the `bbob-largescale` suite and by providing its implementation via the COCO platform.

4.1.3 Automated Benchmarking with the Comparing Continuous Optimizers Platform

The COCO platform [43, 44] has been designed to simplify and standardize the tedious tasks of benchmarking black-box algorithms in continuous domain. It provides several test suites (for example the unconstrained single-objective `bbob` and `bbob-noisy` suites and the bi-objective `bbob-biobj` suite), interfaces several languages (C/C++, Java, Matlab/Octave, Python, R) and supports Linux, Mac, and Windows operating systems. Provided example experiment scripts showcase how to connect basic algorithms to the supported test suites. During an experiment, performance data in terms of runtimes to reach predetermined target function values for each problem instance are automatically collected and written to files. Those data files can then be read in with COCO's postprocessing module (written in Python) that displays performance in graphical and tabular form in both pdf and html format. A great advantage of the standardized COCO data format is that data from a few hundred algorithm variants can by now be compared easily with its postprocessing.

In order to introduce the new `bbob-largescale` test suite in the next section, we will first discuss the basic COCO terminology and philosophy, especially regarding the ideas of problem instances, recorded runtimes, and function target values.

We consider single-objective, unconstrained minimization problems of the form

$$\min_{\mathbf{x}\in\mathbb{R}^n} \; f(\mathbf{x}), \tag{4.3}$$

where n is the problem dimension. The objective is to find, as quickly as possible, one or several solutions $\mathbf{x}$ in the search space $\mathbb{R}^n$ with *small* value(s) $f(\mathbf{x}) \in \mathbb{R}$. We generally measure the *time* of an optimization run as the number

of calls to (queries of) the objective function f.

More precisely, the term objective **function** f refers to a parametrized mapping $\mathbb{R}^n \to \mathbb{R}$, where n is not a priori specified, i.e. the search space is scalable. The parametrization allows the definition of different *instances* of f, by applying transformations in the search or objective space, e.g. rotations or translations.

A **problem** is an instance of an objective function on which the optimization algorithm under consideration is run. Aiming to assess the performance of the algorithm, we further attach target f-values to the problem.

The measure that is used to evaluate the algorithm's performance is the **runtime**, or **run-length**, defined as the conducted *number of evaluations*, also referred to as number of *function* evaluations, to reach a given target on a given problem for the first time. These targets are determined by a set of fixed target precisions added to the optimal f-value.

Collecting such problems constitutes the **test-** or **benchmark-suite**.

Functions, Instances and Problems

Each function in a COCO suite is defined and *parametrized* by the (input) dimension, $n \in \mathbb{N}_+$, its identifier $i \in \mathbb{N}_+$, and the instance number, $j \in \mathbb{N}_+$, that is:

$$f_i^j \equiv f[n, i, j] : \mathbb{R}^n \to \mathbb{R}, \quad \mathbf{x} \mapsto f_i^j(\mathbf{x}) = f[n, i, j](\mathbf{x}). \tag{4.4}$$

In the previous context, a fixed triple $[n, i, j] \equiv [n, f_i, j]$ corresponds to the optimization problem presented to the optimization algorithm. Diversifying n or j varies the search space dimension or the instance respectively of the same objective function $i \equiv f_i$.

Specific instances are deterministically defined as specific sets of transformations applied to the objective function. The instance number j is in practice the integer that is used for seeding the pseudo-random generation of the transformations.

One advantage of problem instances in a test suite is that experiments of algorithms on slightly varying instances of the same underlying function allows to naturally compare stochastic with deterministic algorithms. The recorded runtimes over the instances of a function can be interpreted (for both stochastic and deterministic algorithms) in the same way as runtimes from multiple runs on the same instance of a stochastic algorithm.

Runtime and Target Values

In order to measure the runtime (number of function evaluations) of an algorithm on a problem, we prescribe a **target f-value**, t [41]. In a single run, if the target value t of a problem (f_i, n, j, t) is reached or surpassed, the problem

is solved.[3][4] Recorded runtimes are the only means of evaluating the algorithm performance. Runtimes can be quantitatively interpreted on a ratio scale and allow to measure scaling with the dimension. They are undetermined if the problem is not solved in a single run—however lower bounded by the total number of f-evaluations of this run. Since larger budgets increase the probability of reaching the targets, they are generally preferable. Reasonable termination conditions are not to be disregarded, though, and restarts should be conducted in case [42].

4.1.4 The `bbob-largescale` Test Suite

The `bbob-largescale` test suite provides 24 functions in six dimensions (20, 40, 80, 160, 320 and 640) within the COCO framework. All 24 functions are, in principle, scalable to an arbitrary dimension. The suite is derived from the existing single-objective, unconstrained `bbob` test suite with modifications that allow the user to benchmark algorithms on higher-dimensional problems efficiently. As the experimental setup for the `bbob` suite specifies dimensions 2, 3, 5, 10, 20, and optionally also dimension 40, a natural extension was to use dimension 40 or 80 as the smallest dimension in the new suite. However, in order to facilitate comparison and verification across both test suites, we decide to guarantee one overlapping dimension, namely 20. Hence, the `bbob-largescale` suite starts with dimension 20 and provides, following the tried-and-tested setting for the `bbob` testbed, six different dimensions, increasing by a factor of two up to dimension 640, where the last dimension is again optional. Based on the current implementation of the functions, it is however straightforward to adapt the suite implementation to any set of dimensions, in particular to even larger dimensions. We explain in this section how the `bbob-largescale` test suite is built.

The single-objective `bbob` functions

The `bbob` test suite relies on the use of so-called *raw* functions from which 24 `bbob` functions are generated. A series of transformations on these raw functions, such as linear transformations (e.g., translation, rotation, scaling) and/or non-linear transformations (e.g., $T_{\text{osz}}, T_{\text{asy}}$) is applied to obtain the actual `bbob` test functions. For example, the test function $f_{13}(\mathbf{x})$ (Sharp Ridge function) with (vector) variable $\mathbf{x}$ is derived from a raw function defined as follows:

$$f_{\text{raw}}^{\text{Sharp Ridge}}(\mathbf{z}) = z_1^2 + 100\sqrt{\sum_{i=2}^{n} z_i^2}.$$

Then one applies a sequence of transformations: a translation by using the vector $\mathbf{x}^{\text{opt}}$; then a rotational transformation $\mathbf{R}$; then a scaling transformation $\mathbf{\Lambda}^{10}$; then another rotational transformation $\mathbf{Q}$ to get the relationship

[3] Note that we use the term *problem* in two meanings: the tuple (f_i, n, j) is the concrete objective function, an algorithm $\mathcal{A}$ has access to while in combination with a target t, we are interested in the runtime $\mathrm{RT}(f_i, n, j, t)$ of $\mathcal{A}$ to hit the target t (which might fail). Each problem (f_i, n, j) gives raise to a collection of dependent problems (f_i, n, j, t). Viewed as random variables, $\mathrm{RT}(f_i, n, j, t)$ given (f_i, n, j) are not independent for different values of t.

[4] Target values are directly linked to a problem, leaving the burden to properly define the targets with the designer of the benchmark suite. The alternative is to present final f-values as results, leaving the (rather unsurmountable) burden to interpret these values to the reader. Fortunately, there is an automatized generic way to generate target values from observed runtimes, the so-called run-length based target values [41].

$\mathbf{z} = \mathbf{Q}\boldsymbol{\Lambda}^{10}\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}})$; and finally a translation in objective space by using $\mathbf{f}_{\text{opt}}$ to obtain the final function in the testbed:

$$f_{13}(\mathbf{x}) = f_{\text{raw}}^{\text{Sharp Ridge}}(\mathbf{z}) + \mathbf{f}_{\text{opt}}.$$

There are two main reasons behind the use of transformations here:

(i) provide non-trivial problems that cannot be solved by simply exploiting some of their properties (separability, optimum at fixed position, ...) and

(ii) allow to generate different instances, ideally of similar difficulty, of the same problem by using different (pseudo) random transformations.

Rotational transformations are used to avoid separability and thus coordinate system dependence in the test functions. The rotational transformations consist in applying an orthogonal matrix to the search space: $\mathbf{x} \mapsto \mathbf{z} = \mathbf{R}\mathbf{x}$, where $\mathbf{R}$ is the orthogonal matrix. While the other transformations used in the `bbob` test suite could be naturally extended to the large-scale setting due to their linear complexity, rotational transformations have quadratic time and space complexities. Thus, we need to reduce the complexity of these transformations in order for them to be usable, in practice, in the large-scale setting.

Extension to large-scale setting

Our objective is to construct a large-scale test suite where the cost of a function call is acceptable in higher dimensions while preserving the main characteristics of the original functions in the `bbob` test suite. To this end, we replace the dense orthogonal matrices of the rotational transformations with orthogonal transformations that have linear complexity in the problem dimension: *permuted orthogonal block-diagonal matrices* [4].

Specifically, the matrix of a rotational transformation $\mathbf{R}$ is represented as:

$$\mathbf{R} = \mathbf{P}_{\text{left}}\mathbf{B}\mathbf{P}_{\text{right}}. \tag{4.5}$$

Here, $\mathbf{P}_{\text{left}}$ and $\mathbf{P}_{\text{right}}$ are two permutation matrices[5] and $\mathbf{B}$ is a block-diagonal matrix of the form:

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_1 & 0 & \dots & 0 \\ 0 & \mathbf{B}_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & \mathbf{B}_{n_b} \end{pmatrix}, \tag{4.6}$$

[5] A *permutation matrix* is a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere.

where n_b is the number of blocks and $\mathbf{B}_i, 1 \leq i \leq n_b$ are square matrices of sizes $s_i \times s_i$ satisfying $s_i \geq 1$ and $\sum_{i=1}^{n_b} s_i = n$. If we choose the matrices $\mathbf{B}_i, 1 \leq i \leq n_b$ such that they are all orthogonal, the resulting matrix $\mathbf{B}$ is also an orthogonal matrix. In the `bbob-largescale` test suite, we set $s_i = s := \min(n, 40)$ for all $i = 1, 2, ..., n_b$ (except for the last block which can be smaller)[6] and thus $n_b = \lceil n/s \rceil$.

This representation allows the rotational transformation $\mathbf{R}$ to satisfy three desired properties:

1. Have (almost) linear cost (due to the block structure of $\mathbf{B}$).
2. Introduce non-separability.
3. Preserve the eigenvalues and therefore the condition number of the original function when it is convex quadratic (since $\mathbf{R}$ is orthogonal).

We refer to [4] for all the details of generating the orthogonal block matrices and the permutation matrices involved to the search space transformations.

Adjustments of the functions for scalability performance assessments

Apart from the important modification of the applied rotational transformations described above, which aims at reducing the computational cost of evaluating the function values, further adjustments of the test suite's function definitions are made in order to compare the performance of algorithms with increasing dimensions in a correct way.

The goal of these adjustments is twofold. First, the intrinsic difficulty of the test functions should be independent of the dimension. Second, the range of target values should be defined compatible with how the performance is assessed within the COCO framework. Since this is achieved by recording the same target precision values over all problems (fixed within a given range), the function values are rescaled for each function to avoid that target precisions become too easy to reach when the dimension increases. Without this adjustment, even very simple algorithms such as the pure random search may be able to solve a relevant proportion of some test problems, leading to misinterpretations of algorithm performances.

In particular, we made the following three changes to the raw functions in the `bbob` test suite.

- All functions are normalized by dimension. Except for the six functions Schwefel, Schaffer, Weierstrass, Gallagher, Griewank-Rosenbrock and Katsuura, which are already normalized with dimension, the functions are normalized by the parameter $\gamma(n) = \min(1, 40/n)$ to make their target values comparable, in difficulty, over a wide range of dimensions without losing backwards compatibility.
- The Discus, Bent Cigar and Sharp Ridge functions are generalized such that they have a constant proportion of $\lceil n/40 \rceil$ distinct axes that remain consistent with the `bbob` test suite.

[6] This setting allows to have the problems in dimensions 20 and 40 overlap between the `bbob` test suite and its large-scale extension since in these dimensions, the block sizes coincide with the problem dimensions.

Table 4.1: Function descriptions of the separable and moderately conditioned function groups of the `bbob-largescale` test suite.

Group 1: Separable functions

Formulation	*Transformations*
Sphere Function $f_1(\mathbf{x}) = \gamma(n) \times \sum_{i=1}^{n} z_i^2 + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{x} - \mathbf{x}^{\text{opt}}$
Ellipsoidal Function $f_2(\mathbf{x}) = \gamma(n) \times \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} z_i^2 + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = T_{\text{osz}}\left(\mathbf{x} - \mathbf{x}^{\text{opt}}\right)$
Rastrigin Function $f_3(\mathbf{x}) = \gamma(n) \times \left(10n - 10\sum_{i=1}^{n} \cos(2\pi z_i) + \|z\|_2^2\right) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{\Lambda}^{10} T_{\text{asy}}^{0.2}\left(T_{\text{osz}}\left(\mathbf{x} - \mathbf{x}^{\text{opt}}\right)\right)$
Bueche-Rastrigin Function $f_4(\mathbf{x}) = \gamma(n) \times \left(10n - 10\sum_{i=1}^{n} \cos(2\pi z_i) + \|z\|_2^2\right) + 100 f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$z_i = s_i T_{\text{osz}}\left(x_i - x_i^{\text{opt}}\right)$ for $i = 1, \ldots, n$, $s_i = \begin{cases} 10 \times 10^{\frac{1}{2}\frac{i-1}{n-1}} & \text{if } z_i > 0 \text{ and } i \text{ odd} \\ 10^{\frac{1}{2}\frac{i-1}{n-1}} & \text{otherwise} \end{cases}$ for $i = 1, \ldots, n$
Linear Slope Function $f_5(\mathbf{x}) = \gamma(n) \times \sum_{i=1}^{n} \left(5\lvert s_i\rvert - s_i z_i\right) + \mathbf{f}_{\text{opt}}$	$z_i = \begin{cases} x_i & \text{if } x_i^{\text{opt}} x_i < 5^2 \\ x_i^{\text{opt}} & \text{otherwise} \end{cases}$ for $i = 1, \ldots, n$, $s_i = \text{sign}\left(x_i^{\text{opt}}\right) 10^{\frac{i-1}{n-1}}$ for $i = 1, \ldots, n$, $\mathbf{x}^{\text{opt}} = \mathbf{z}^{\text{opt}} = 5 \times \mathbf{1}_-^+$

Group 2: Functions with low or moderate conditioning

Attractive Sector Function $f_6(\mathbf{x}) = T_{\text{osz}}\left(\gamma(n) \times \sum_{i=1}^{n} \left(s_i z_i\right)^2\right)^{0.9} + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{Q}\mathbf{\Lambda}^{10}\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}})$ with $\mathbf{R} = P_{11}B_1P_{12}, \mathbf{Q} = P_{21}B_2P_{22}$, $s_i = \begin{cases} 10^2 & \text{if } z_i \times x_i^{\text{opt}} > 0 \\ 1 & \text{otherwise} \end{cases}$ for $i = 1, \ldots, n$
Step Ellipsoidal Function $f_7(\mathbf{x}) = \gamma(n) \times 0.1 \max\left(\lvert\hat{z}_1\rvert/10^4, \sum_{i=1}^{n} 10^{2\frac{i-1}{n-1}} z_i^2\right) + f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$\hat{\mathbf{z}} = \mathbf{\Lambda}^{10}\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}})$ with $\mathbf{R} = P_{11}B_1P_{12}$, $\tilde{z}_i = \begin{cases} \lfloor 0.5 + \hat{z}_i \rfloor & \text{if } \lvert\hat{z}_i\rvert > 0.5 \\ \lfloor 0.5 + 10\hat{z}_i \rfloor/10 & \text{otherwise} \end{cases}$ for $i = 1, \ldots, n$, $\mathbf{z} = \mathbf{Q}\tilde{\mathbf{z}}$ with $\mathbf{Q} = P_{21}B_2P_{22}$
Rosenbrock Function, original $f_8(\mathbf{x}) = \gamma(n) \times \sum_{i=1}^{n-1} \left(100\left(z_i^2 - z_{i+1}\right)^2 + \left(z_i - 1\right)^2\right) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \max\left(1, \frac{\sqrt{s}}{8}\right)(\mathbf{x} - \mathbf{x}^{\text{opt}}) + \mathbf{1}$, $\mathbf{x}^{\text{opt}} \in [-3, 3]^n$
Rosenbrock Function, rotated $f_9(\mathbf{x}) = \gamma(n) \times \sum_{i=1}^{n-1} \left(100\left(z_i^2 - z_{i+1}\right)^2 + \left(z_i - 1\right)^2\right) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \max\left(1, \frac{\sqrt{s}}{8}\right)\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}}) + \mathbf{1}$ with $\mathbf{R} = P_1BP_2$, $\mathbf{x}^{\text{opt}} \in [-3, 3]^n$

- For the two Rosenbrock functions and the related Griewank-Rosenbrock function, a different scaling is used than in the original `bbob` functions: instead of using the factor $\max(1, \frac{\sqrt{n}}{8})$ with n being the problem dimension, we scale the rotated search vector by the factor $\max(1, \frac{\sqrt{s}}{8})$, where $s = \min(n, 40)$ is the block size in the matrix B. Because $\sqrt{40} < 8$, this corresponds to no scaling. An additional constant is added to the z vector to reduce, with high probability, the risk to move important parts of the test function's characteristics out of the domain of interest. Without these adjustments, the original functions become significantly easier in higher dimensions due to the optimum being too close to the origin. For more details, we refer the interested reader to the discussion on the corresponding GitHub issue [1].

For a better understanding of the properties of these functions and for the definitions of the used transformations and abbreviations, we refer the reader to the original `bbob` function documentation [84].

Functions in the Suite

Tables 4.1, 4.2, and 4.3 below present the definition of all 24 functions of the `bbob-largescale` test suite in detail.

Table 4.2: Function descriptions of the ill-conditioned and adequately structured multimodal function groups of the bbob-largescale test suite.

Group 3: Ill-conditioned functions

Formulation	*Transformations*
Ellipsoidal Function $f_{10}(\mathbf{x}) = \gamma(n) \times \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} z_i^2 + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = T_{\text{osz}}(\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}}))$ with $\mathbf{R} = P_1 B P_2$
Discus Function $f_{11}(\mathbf{x}) = \gamma(n) \times \left(10^6 \sum_{i=1}^{\lceil n/40 \rceil} z_i^2 + \sum_{i=\lceil n/40 \rceil+1}^{n} z_i^2\right) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = T_{\text{osz}}(\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}}))$ with $\mathbf{R} = P_1 B P_2$
Bent Cigar Function $f_{12}(\mathbf{x}) = \gamma(n) \times \left(\sum_{i=1}^{\lceil n/40 \rceil} z_i^2 + 10^6 \sum_{i=\lceil n/40 \rceil+1}^{n} z_i^2\right) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{R} T_{\text{asy}}^{0.5}(\mathbf{R}((\mathbf{x} - \mathbf{x}^{\text{opt}}))$ with $\mathbf{R} = P_1 B P_2$
Sharp Ridge Function $f_{13}(\mathbf{x}) = \gamma(n) \times \left(\sum_{i=1}^{\lceil n/40 \rceil} z_i^2 + 100\sqrt{\sum_{i=\lceil n/40 \rceil+1}^{n} z_i^2}\right) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{Q}\mathbf{\Lambda}^{10}\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}})$ with $\mathbf{R} = P_{11} B_1 P_{12}, \mathbf{Q} = P_{21} B_2 P_{22}$
Different Powers Function $f_{14}(\mathbf{x}) = \gamma(n) \times \sum_{i=1}^{n} \lvert z_i \rvert^{\left(2+4\times\frac{i-1}{n-1}\right)} + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}})$ with $\mathbf{R} = P_1 B P_2$

Group 4: Multi-modal functions with adequate global structure

Rastrigin Function $f_{15}(\mathbf{x}) = \gamma(n) \times \left(10n - 10\sum_{i=1}^{n} \cos(2\pi z_i) + \lVert\mathbf{z}\rVert_2^2\right) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{R}\mathbf{\Lambda}^{10}\mathbf{Q} T_{\text{asy}}^{0.2}\left(T_{\text{osz}}\left(\mathbf{R}\left(\mathbf{x} - \mathbf{x}^{\text{opt}}\right)\right)\right)$ with $\mathbf{R} = P_{11} B_1 P_{12}, \mathbf{Q} = P_{21} B_2 P_{22}$
Weierstrass Function $f_{16}(\mathbf{x}) = 10\left(\frac{1}{n}\sum_{i=1}^{n}\sum_{k=0}^{11} \frac{1}{2^k} \cos\left(2\pi 3^k (z_i + 1/2)\right) - f_0\right)^3 + \frac{10}{n} f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{R}\mathbf{\Lambda}^{1/100}\mathbf{Q} T_{\text{osz}}(\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}}))$ with $\mathbf{R} = P_{11} B_1 P_{12}, \mathbf{Q} = P_{21} B_2 P_{22}$, $f_0 = \sum_{k=0}^{11} \frac{1}{2^k} \cos(\pi 3^k)$
Schaffers F7 Function $f_{17}(\mathbf{x}) = \left(\frac{1}{n-1}\sum_{i=1}^{n-1}\left(\sqrt{s_i} + \sqrt{s_i}\sin^2\left(50(s_i)^{1/5}\right)\right)\right)^2 + 10 f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{\Lambda}^{10}\mathbf{Q} T_{\text{asy}}^{0.5}(\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}}))$ with $\mathbf{R} = P_{11} B_1 P_{12}, \mathbf{Q} = P_{21} B_2 P_{22}$ $s_i = \sqrt{z_i^2 + z_{i+1}^2}, i = 1, \ldots, n-1$
Schaffers F7 Function, moderately ill-conditioned $f_{18}(\mathbf{x}) = \left(\frac{1}{n-1}\sum_{i=1}^{n-1}\left(\sqrt{s_i} + \sqrt{s_i}\sin^2\left(50(s_i)^{1/5}\right)\right)\right)^2 + 10 f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \mathbf{\Lambda}^{1000}\mathbf{Q} T_{\text{asy}}^{0.5}(\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}}))$ with $\mathbf{R} = P_{11} B_1 P_{12}, \mathbf{Q} = P_{21} B_2 P_{22}$, $s_i = \sqrt{z_i^2 + z_{i+1}^2}, i = 1, \ldots, n-1$
Composite Griewank-Rosenbrock Function F8F2 $f_{19}(\mathbf{x}) = \frac{10}{n-1}\sum_{i=1}^{n-1}\left(\frac{s_i}{4000} - \cos(s_i)\right) + 10 + \mathbf{f}_{\text{opt}}$	$\mathbf{z} = \max\left(1, \frac{\sqrt{s}}{8}\right)\mathbf{R}\mathbf{x} + \frac{\mathbf{1}}{2}$ with $\mathbf{R} = P_1 B P_2$, $s_i = 100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2$, for $i = 1, \ldots, n-1$, $\mathbf{z}^{\text{opt}} = \mathbf{1}$

4.1.5 Implementation of the large-scale testbed and repository for datasets

The bbob-largescale suite is implemented within the COCO open source project and the code is available in the repository github.com/numbbo/coco. Its test problems are implemented in C based on the COCO problem structure coco_problem_s. One main purpose of the COCO platform is to attract researchers from various domains of continuous optimization to assess and compare the performance of their algorithms in a generic black-box setting. Any researcher can provide datasets of benchmarked solvers, which are collected in a publicly available repository and are directly available for comparison with any other solver. Historically, this collection of datasets has been performed through the Black-Box Optimization Benchmarking (BBOB) workshop series. For the bbob-largescale test suite, 11 data sets are already available online. In 4.1.6 we provide a detailed guide on using the COCO platform and in particular the bbob-largescale suite, as well as accessing and post processing the datasets collected in the past.

The bbob-largescale suite has been used in [97] (see section 2.3) to analyze the search performance of large-scale CMA-ES [37] variants and it is an example of how the proposed suite allows the differentiation among algorithms.[7] However, no details about the used test problems were provided.

[7]The same post processed data with [97] are used in the guide of 4.1.7, as output example of COCO, where it is clarified how the platform allows the algorithm differentiation and which scientific information we can obtain from the benchmarking procedure.

Table 4.3: Function descriptions of the ill-conditioned and adequately structured multimodal function groups of the `bbob-largescale` test suite.

Group 5: Multi-modal functions with weak global structure

Formulation	*Transformations*
Schwefel Function $f_{20}(\mathbf{x}) = -\frac{1}{100n}\sum_{i=1}^{n} z_i \sin\left(\sqrt{\lvert z_i\rvert}\right) + 4.189828872724339 + 100 f_{pen}(\mathbf{z}/100) + \mathbf{f}_{\text{opt}}$	$\hat{\mathbf{x}} = 2 \times \mathbf{1}^+_- \otimes \mathbf{x}$, $\hat{z}_1 = \hat{x}_1$, $\hat{z}_{i+1} = \hat{x}_{i+1} + 0.25\left(\hat{x}_i - 2\left\lvert x_i^{\text{opt}}\right\rvert\right)$, for $i = 1, \ldots, n-1$, $\mathbf{z} = 100\left(\boldsymbol{\Lambda}^{10}\left(\hat{\mathbf{z}} - 2\left\lvert\mathbf{x}^{\text{opt}}\right\rvert\right) + 2\left\lvert\mathbf{x}^{\text{opt}}\right\rvert\right)$, $\mathbf{x}^{\text{opt}} = 4.2096874633/2\mathbf{1}^+_-$
Gallagher's Gaussian 101-me Peaks Function $f_{21}(\mathbf{x}) = T_{\text{osz}}\left(10 - \max_{i=1}^{101}\left(w_i \exp\left(-\frac{1}{2n}(\mathbf{z}-\mathbf{y}_i)^T\, \mathbf{B}^T\mathbf{C_i}\mathbf{B}(\mathbf{z}-\mathbf{y}_i)\right)\right)\right)^2 + f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$w_i = \begin{cases} 1.1 + 8 \times \frac{i-2}{99} & \text{for } 2 \le i \le 101 \\ 10 & \text{for } i = 1 \end{cases}$ $\mathbf{B}$ is a block-diagonal matrix without permutations of the variables. $\mathbf{C_i} = \Lambda^{\alpha_i}/\alpha_i^{1/4}$,where Λ^{α_i} is defined as usual, but with randomly permuted diagonal elements. For $i = 2, \ldots, 101$, α_i is drawn uniformly from the set $\left\{1000^{2\frac{j}{99}}, j = 0, \ldots, 99\right\}$ without replacement, and $\alpha_i = 1000$ for $i = 1$. The local optima $\mathbf{y}_i$ are uniformly drawn from the domain $[-5,5]^n$ for $i = 2, \ldots, 101$ and $\mathbf{y}_1 \in [-4,4]^n$. The global optimum is at $\mathbf{x}^{\text{opt}} = \mathbf{y}_1$.
Gallagher's Gaussian 21-hi Peaks Function $f_{22}(\mathbf{x}) = T_{\text{osz}}\left(10 - \max_{i=1}^{21}\left(w_i \exp\left(-\frac{1}{2n}(\mathbf{z}-\mathbf{y}_i)^T\, \mathbf{B}^T\mathbf{C_i}\mathbf{B}(\mathbf{z}-\mathbf{y}_i)\right)\right)\right)^2 + f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$w_i = \begin{cases} 1.1 + 8 \times \frac{i-2}{19} & \text{for } 2 \le i \le 21 \\ 10 & \text{for } i = 1 \end{cases}$ $\mathbf{B}$ is a block-diagonal matrix without permutations of the variables. $\mathbf{C_i} = \Lambda^{\alpha_i}/\alpha_i^{1/4}$, where Λ^{α_i} is defined as usual, but with randomly permuted diagonal elements. For $i = 2, \ldots, 21$, α_i is drawn uniformly from the set $\left\{1000^{2\frac{j}{19}}, j = 0, \ldots, 19\right\}$ without replacement, and $\alpha_i = 1000^2$ for $i = 1$. The local optima $\mathbf{y}_i$ are uniformly drawn from the domain $[-4.9, 4.9]^n$ for $i = 2, \ldots, 21$ and $\mathbf{y}_1 \in [-3.92, 3.92]^n$. The global optimum is at $\mathbf{x}^{\text{opt}} = \mathbf{y}_1$.
Katsuura Function $f_{23}(\mathbf{x}) = f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}} + \left(\frac{10}{n^2}\prod_{i=1}^{n}\left(1 + i\sum_{j=1}^{32}\frac{\lvert 2^j z_i - [2^j z_i]\rvert}{2^j}\right)^{10/n^{1.2}} - \frac{10}{n^2}\right)$	$\mathbf{z} = \mathbf{Q}\boldsymbol{\Lambda}^{100}\mathbf{R}(\mathbf{x} - \mathbf{x}^{\text{opt}})$ with $\mathbf{R} = P_{11}B_1P_{12}$, $\mathbf{Q} = P_{21}B_2P_{22}$
Lunacek bi-Rastrigin Function $f_{24}(\mathbf{x}) = \gamma(n) \times \left(\min\left(\sum_{i=1}^{n}(\hat{x}_i - \mu_0)^2, n + s\sum_{i=1}^{n}(\hat{x}_i - \mu_1)^2\right) + 10\left(n - \sum_{i=1}^{n}\cos(2\pi z_i)\right)\right) + 10^4 f_{pen}(\mathbf{x}) + \mathbf{f}_{\text{opt}}$	$\hat{\mathbf{x}} = 2\text{sign}(\mathbf{x}^{\text{opt}}) \otimes \mathbf{x}$, $\mathbf{x}^{\text{opt}} = 0.5\mu_0\mathbf{1}^+_-$ $\mathbf{z} = \mathbf{Q}\boldsymbol{\Lambda}^{100}\mathbf{R}(\hat{\mathbf{x}} - \mu_0\mathbf{1})$with $\mathbf{R} = P_{11}B_1P_{12}$, $\mathbf{Q} = P_{21}B_2P_{22}$, $\mu_0 = 2.5, \mu_1 = -\sqrt{\frac{\mu_0^2 - 1}{s}}$, $s = 1 - \frac{1}{2\sqrt{n+20} - 8.2}$

4.1.6 A guide for benchmarking with COCO

The code basis of COCO consists of two parts:

The experiments part It defines the test suites, allows to conduct the experiments and provides the output data to be postprocessed. The code is written in C and wrapped in other languages (currently C/C++, Java, Matlab/Octave and Python), providing an easy-to-use interface. Apart from the currently implemented test suites, COCO allows the definition and integration of new test problems, as well as other functionalities, e.g. data logging options.

The Postprocessing It processes the output data from the experimental part, provides the option of processing data from previously archived datasets, and generates various figures and tables presenting aggregated runtime results.

Launching experiments

For the installation steps, we refer to the Getting Started guide of COCO [2]. After installation, launching an experiment slightly differs for each language. The `example_experiment` file is modified so that the solver to be benchmarked is connected to COCO and other parameters of the experiment are set. In Python, for which the more recent `example_experiment2.py` file is available, the following additions and modifications compared to the default choices are left to the user:

(i) The necessary imports and the definition of the desired optimizer to be benchmarked:

```
import scipy.optimize
fmin = scipy.optimize.fmin_l_bfgs_b
```

(ii) The selection of the test suite and the maximum budget of function evaluations:

```
suite_name = "bbob-largescale"
budget_multiplier = 1e4  # times dimension, increase to 10, 100, ...
```

The maximum number of function evaluations on each problem equals to the budget multiplier times the problem dimension. It is highly advisable to run the first experiments with a much smaller budget multiplier, for example 2, 5, or 10.

(iii) The user can optionally filter the suite and perform the experiment on a subset of the suite problems. For example, one can exclude the largest dimension 640 and select specific problem instances:

```
suite_filter_options = ( "dimensions: 20,40,80,160,320 " +
                         "instance_indices: 1-5 ")
```

(iv) In Python, an automatized way for a parallel execution of the experiment is provided: running the experiment in batches generates a partition of the set of problems of the *filtered, as described above,* suite, and the experiment can be performed in parallel for every batch. The execution time of the experiment can be restrictive, e.g., with a large maximum budget or when high-dimensional problems are benchmarked. Setting:

```
batches = 1
current_batch = 10
```

conducts the experiment only on the first out of ten batches.

(v) Finally, the minimizer has to be added in the restarts loop, where the user can set its specific options, e.g., termination conditions. Stopping information can also be recorded:

```
while evalsleft() > 0 and not problem.final_target_hit:
    irestart += 1
    if fmin is scipy.optimize.fmin_l_bfgs_b:
        output = fmin(problem, propose_x0(), approx_grad=True,
                      maxfun=evalsleft())
        stoppings[problem.index].append(output[2]['task'])
```

Many of the options for the experimental setting can also be directly set when the code is called from a system shell, like:

```
python example_experiment2.py budget_multiplier=1e4 batch=1/10 suite_name=bbob-largescale
```

With the execution of the experiment for the first time, a root folder called `exdata` is created. A new subfolder in `exdata` is created with each launched experiment and, in Python, its name by default contains the solver name, the module from which the solver was imported, the maximum budget and the test suite name. This subfolder contains all the logged data of the specific experiment to be later read by the postprocessing. In case of parallel execution, several subfolders are created, one per each batch, also with the batch number contained in their names. In this case, a folder containing all these subfolders must later be passed to the postprocessing.

The Python experiment prints a timing summary like the following

```
*** Full experiment done in 0h10:37 ***
Timing summary:
  dimension  median seconds/evaluations
  -------------------------------------
     20       8.2e-06
     40       1.0e-05
     80       1.6e-05
    160       2.7e-05
    320       5.2e-05
    640       1.0e-04
  -------------------------------------
```

here taken on a 2019 Macbook Pro with `budget_multiplier=10` and minimal overhead from the solver. Hence, an experiment over all functions, instances and dimensions with `budget_multiplier=10000` and parallelized over 20 CPUs will take about 10h for the computations of the function evaluations (not accounting for internal solver time). This time requirement is likely to be small compared to the time requirements of the solver.

Practical hint: It is highly recommended to start the experiments with small budgets, before increasing them gradually. Benchmarking data with different budgets can only be postprocessed as data from separate experiments and cannot be merged. However, the idea is to quickly get completed (and independent) data sets for inspection in order to i) track unexpected results indicating a bug in the code early, ii) successively get reliable estimates for the execution time of longer experiments, and iii) be able to inspect chance variations by directly comparing the generated data sets. In addition, the experiment on the `bbob-largescale` test suite can be easily run in parallel

batches.

Postprocessing

This part of the code, written entirely in Python, aggregates the runtime data to generate various figures and tables in html format and include them into LaTeX documents. Both single algorithm results or comparison results of several algorithms are available. Several ways to aggregate the data are used, and each figure is described in the next section.

Initially, the `cocopp` Python package is installed. Then, executing from a Python shell

```
>>> import cocopp
>>> cocopp.main('[-o OUTPUTFOLDER] YOURDATAFOLDER [MORE_DATAFOLDERS]')
```

or from a system shell:

```
python -m cocopp [-o OUTPUTFOLDER] YOURDATAFOLDER [MORE_DATAFOLDERS]
```

will postprocess the logged data contained in any subfolder of the folder arguments. This allows to collect the data from several batches under root folders, e.g. `YOURDATAFOLDER`. Each one of them corresponds to data from **one** solver. Single-algorithm evaluation results are created in case where only `YOURDATAFOLDER` is given as argument and comparison data when `MORE_DATAFOLDERS` are present. By default, if the `OUTPUTFOLDER` is not specified, the post-processed results are stored in a new folder called `ppdata`, and they can be explored from the `ppdata/index.html` file.

Archived data from over 200 algorithms are also provided by COCO for postprocessing, 11 of them on the `bbob-largescale` suite, allowing a comparison of a wide range of solvers benchmarked in the past. For example,

```
>>> cocopp.archives.bbob_largescale('bfgs')
```

lists all available data sets with `'bfgs'` in their name,

```
>>> cocopp.main('bbob-largescale/.*bfgs')
```

generates comparison data for all data sets of the list, and

```
>>> cocopp.main('bfgs!')
```

postprocesses the first data set with `'bfgs'` in its name (though not necessarily from the `bbob-largescale` suite).

Archived and local data can be mixed for postprocessing, e.g.

```
>>> cocopp.main('YOURDATAFOLDER bbob-largescale/2019/LBFGS')
```

The given substring must match a unique data set of the archive. Otherwise, all data sets that match are listed, but none is postprocessed. To display algorithms in the background, the `genericsettings.background` variable can be set as:

```
>>> cocopp.genericsettings.background = {None: ['DataFolder1', 'DataFolder2', ...]}
```

before running the postprocessing where `None` invokes the default background color and linestyle `cocopp.genericsettings.background_default_style`.

For the creation of a single document with the postprocessed results, COCO provides several LaTeX templates that compile the generated tables and figures. For this,

(i) the template with the associated style files must be copied to the directory where the output folder `ppdata` is and

(ii) the template can be (optionally) edited, in particular the algorithm name(s).

4.1.7 The Different COCO Graphs: How to Read Them and What Can Be Learned From Them

In this section, we present various graphs and tables generated by the COCO Postprocessing (version 2.3.3) and we explain how they quantify the performance comparison and how they can be interpreted. The shown data compare large-scale variants of the Covariance Matrix Adaptation Evolution Strategy [37] and of L-BFGS [57] on the `bbob-largescale` test suite [97].

Runtime distribution graphs (ECDF)

With COCO a benchmarking experiment is recorded as a set of number of function evaluations, also called runtimes, to reach (or surpass) some given target function values on each function and in each dimension. It is natural to display the empirical distribution of these recorded runtimes in empirical cumulative distribution functions (ECDF), denoted as *runtime distributions* in the following. Runtime distributions for a single target value are also known as data profiles [64]. The COCO runtime distribution plots differ in three ways from standard data profiles: (i) the target values do not depend on the shown data; (ii) results for *multiple* targets are aggregated in a single distribution graph; (iii) otherwise undefined runtimes of *unsuccessful* trials are generated by simulated restarts.

In general, a runtime distribution or data profile shows the **success rate** on the y-axis, i.e., the proportion of problems solved (in the sense of Section 4.1.3), for any given budget on the x-axis (measured in number of function evaluations divided by dimension, #f-evals/dimension). Considering the y-axis as independent, we read for any given fraction of problems (sorted by their runtime) their maximal runtime on the x-axis. As an example, Figure 4.1 shows such distributions for six algorithms.

The runtime distribution does not correspond to a single trial: aggregation is over runs with independent restarts and on several instances of a function (Figure 4.1 left) or groups of functions (Figure 4.1 right). An important remark here is that domination of one algorithm over another in the distribution graph does not necessarily mean that the former is faster on every single problem, due to the fact that the displayed runtimes are sorted by length and hence

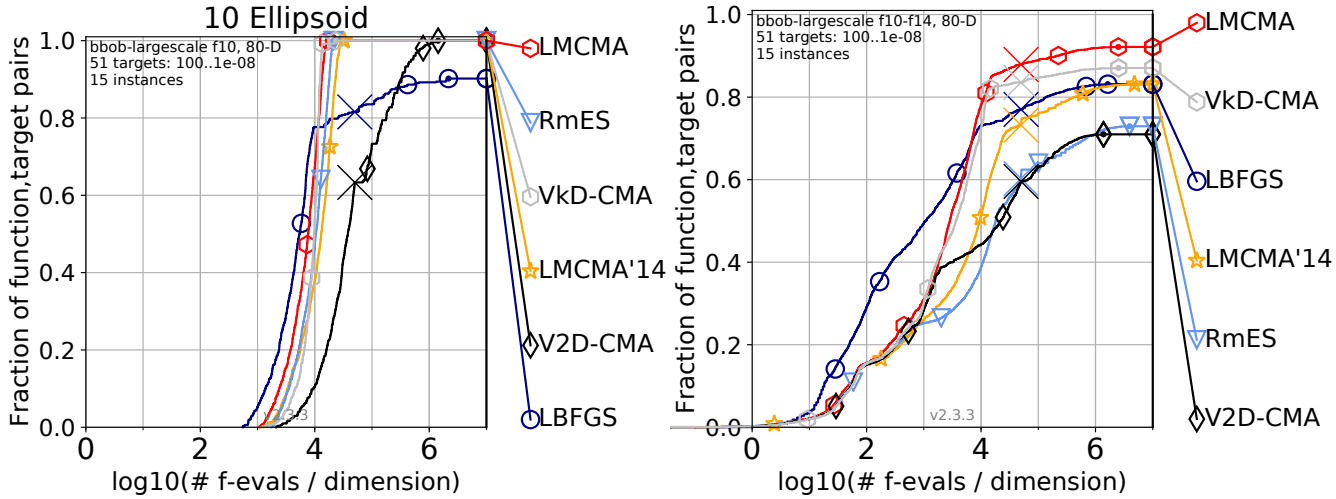

Figure 4.1: Bootstrapped runtime distributions for 51 targets in $10^{[-8..2]}$ for a single function (left) and for the group of functions f10–f14 (right) in dimension 80. f10–f14 is the group of unimodal functions with high conditioning in the `bbob-largescale` suite.

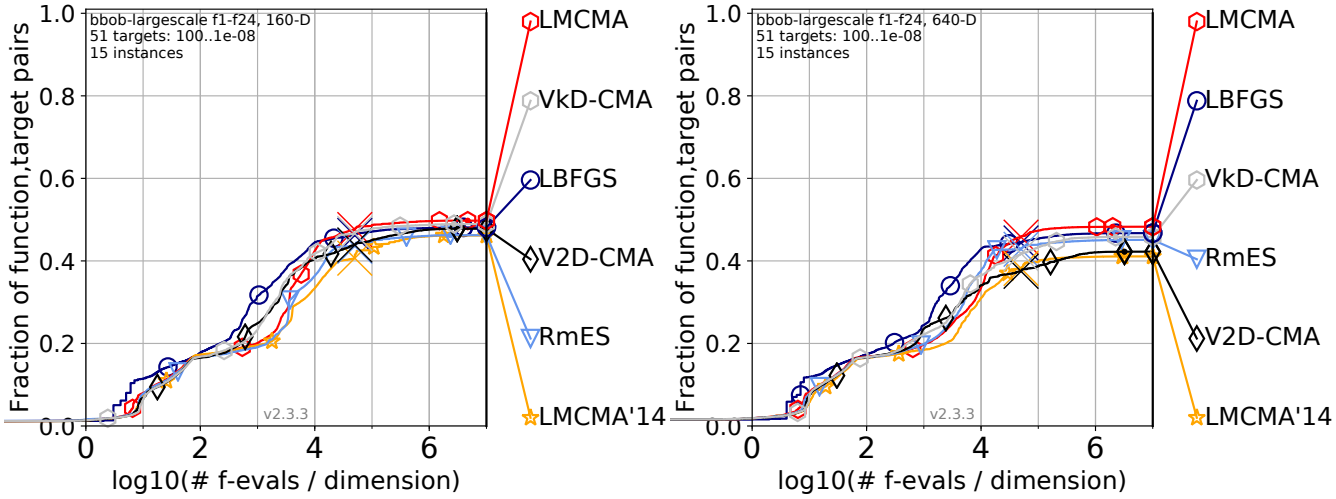

Figure 4.2: Bootstrapped runtime distributions of a variety of largfe-scale solvers, taken from [97]. Shown are 51 targets in $10^{[-8..2]}$ for all functions of the `bbob-largescale` suite in dimension 160 (left) and 640 (right).

differently for each algorithm and the information about the underlying function is lost in the graphs.

If the success ratio on any given problem is smaller than one but greater than zero, the runtime of unsuccessful trials is determined via simulated restarts from the recorded data of all trials on the very same problem (bootstrapped) thereby mimicking the truly restarted algorithm [41].

Runtime distributions allow a quantified comparison between solvers: a horizontal shift of the graph corresponds to a runtime difference with the respective factor. In the figure for the Ellipsoid function, for example, this comparison would be: Limited memory CMA-ES (LMCMA) [59] is $10^{0.2}$ times faster than Rank-m Evolution Strategy (RmES) [56]. They also can expose possible defects of an algorithm: the same figure shows that L-BFGS does not reach the more difficult target values, suggesting that the finite difference approximation of the gradients deteriorates the

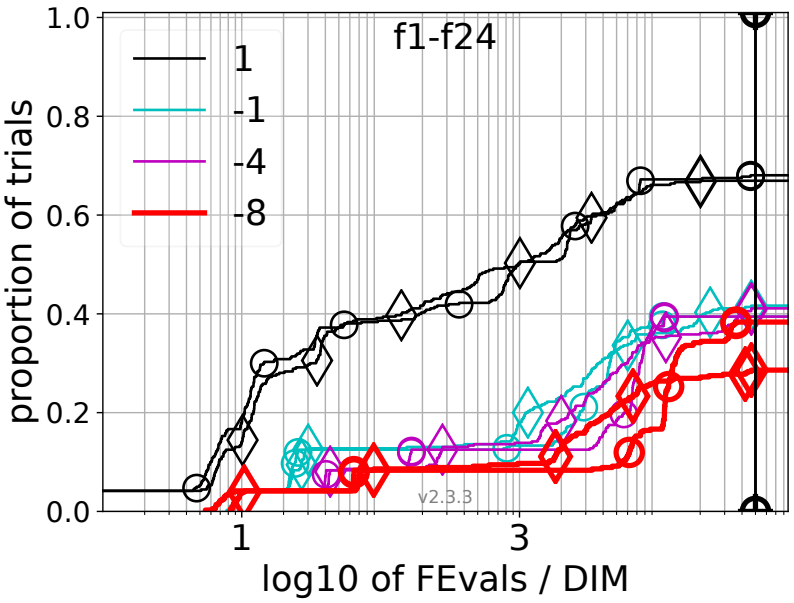

Figure 4.3: Runtime distributions for all functions in dimension 320 to reach a target value $\Delta f + f_{opt}$ with $\Delta f = 10^k$, where k is given in the legend, for LMCMA ($\circ$) and VkD-CMA ($\diamond$).

performance on the ill-conditioned, non separable Ellipsoid function.

A runtime distribution may contain only runtimes to reach a single target value, instead of several ones. In the case of single-solver or two-solvers data, the Postprocessing generates runtime distribution graphs for selected targets and dimensions, where aggregation is over groups of functions (Figures 4.3 and 4.4 left). This way, information for easier problems (larger target values) and more difficult ones for the specific function group is now displayed.

Apart from runtime distributions, other quantities are also considered. In the case of single-algorithm data, the Postprocessing provides distribution graphs of the best achieved target value for given budgets of function evaluations (Figure 4.4 right). In the case of two solvers, runtime **ratio** distributions of the solvers for selected targets are generated (Figure 4.5).

Scaling graphs

In contrast to runtime distributions that display the ECDF of runtimes for different targets (and potentially different functions), a scaling graph like in Figure 4.6 displays the expected (estimated) runtime values (ERT) for a particular function and target value against dimension. As the name indicates, these plots illustrate the scalability of solvers with dimension.

Specifically, the scaling graphs show the expected runtimes to reach a certain target function value which are computed as the sum of all function evaluations of the unsuccessful trials, plus the sum of runtimes until the target is hit of successful trials, both divided by the number of successful trials [41].[8]

The ERT values in #f-evals/dimension are plotted versus dimension in a log-log plot, thus a constant graph

[8] If all trials are successful this is the average runtime.

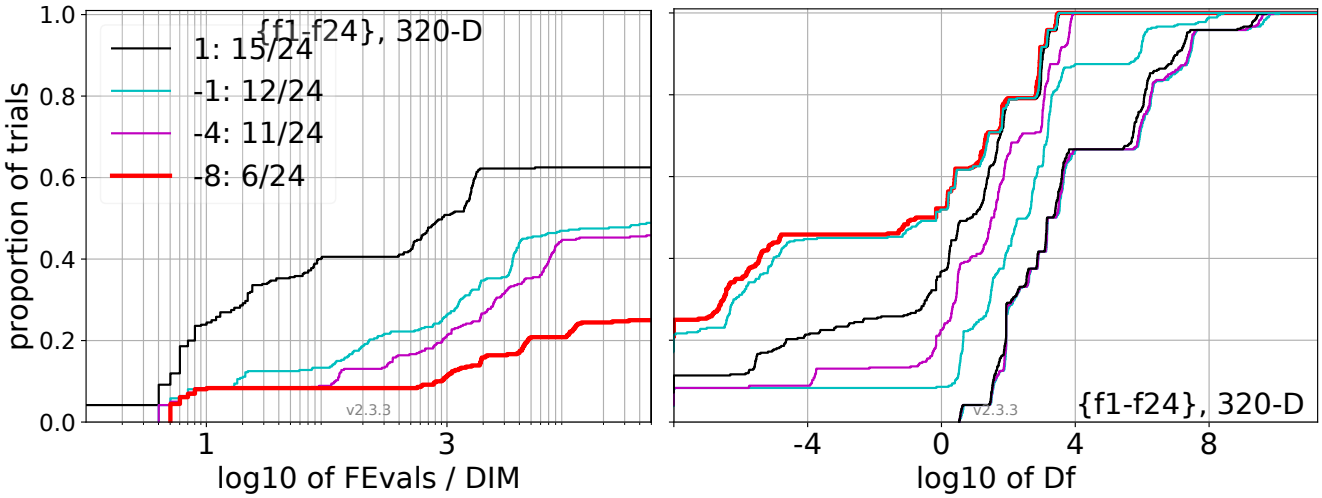

Figure 4.4: Left: ECDF of the number of function evaluations of LBFGS divided by search space dimension, to fall below $f_{\text{opt}} + 10^k$, where k is the first value in the legend. Right: ECDF of the best achieved target value Δf (shown as Df in the axis label) for budgets of 0.5D, 1.2D, 3D, 10D, 100D, ... function evaluations (from right to left cycling cyan-magenta-black ...) and for the total budget (red).

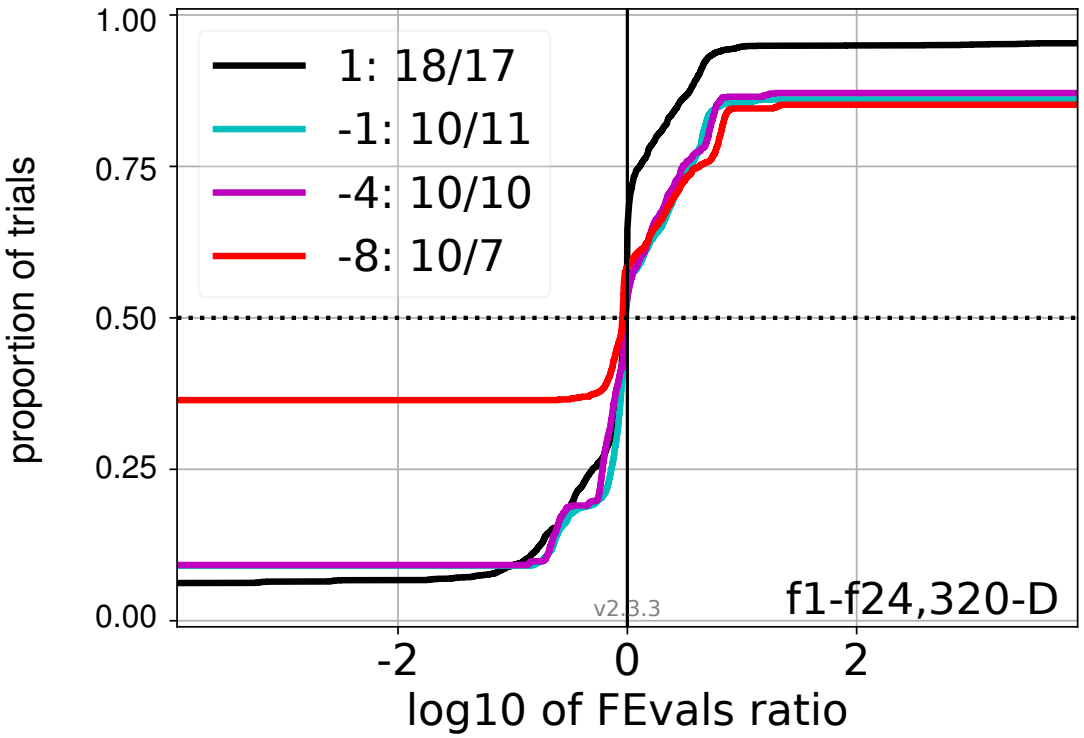

Figure 4.5: ECDF of runtime ratios of LMCMA divided by VkD-CMA for all functions in dimension 320 to reach target values 10^k with k given in the legend; all trial pairs for each function. Pairs where both trials failed are disregarded, pairs where one trial failed are visible in the limits being > 0 or < 1. The legend also indicates, after the colon, the number of functions that were solved in at least one trial (LMCMA first).

corresponds to linear scaling. Slanted grid lines indicate quadratic scaling.

Figure 4.6 shows the scaling of CMA-ES variants and L-BFGS on the linear slope function. It is linear for most solvers, except for those with a population size larger than the default (solvers with suffices P2 and P10). Specifically

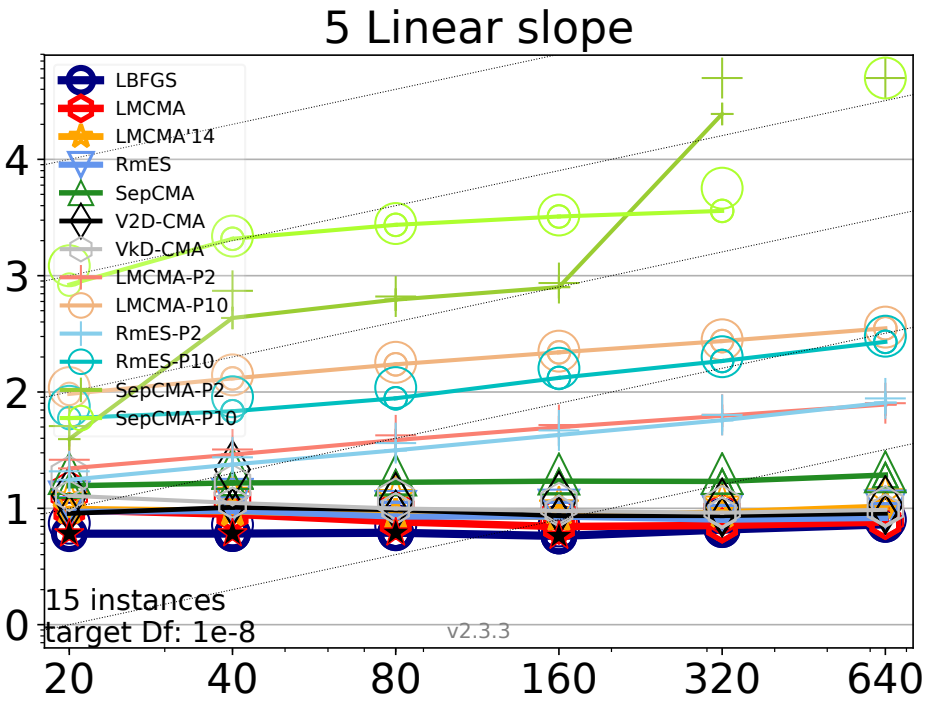

Figure 4.6: Expected running time (ERT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8}. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction by the number of dimensions (six).

for the separable CMA-ES with larger population sizes, the graphs reveal a performance defect in particular in larger dimension due to the step size adaptation mechanism, as verified after supplementary experiments, see also [97].

Scatter plots

In the case of comparison of two solvers, the COCO postprocessing generates *scatter plots* of the algorithms' ERT values for several targets for every function of the suite, see Figure 4.7 for an example. The graph is in log-log scale and the first solver corresponds to the y-axis. Each color represents a different dimension.

Scatter plots maintain information for single problems separately (after averaging over instances), since for every function and for every target the average runtime is displayed, allowing a comparison between easier and more difficult problems.

Figure 4.7 illustrates that on the Ellipsoid function only in dimensions smaller than 80 VkD-CMA (k Vectors and Diagonal Covariance Matrix Evolution Strategy, [6]) outperforms LMCMA on the difficult target values, by a factor increasing with the target value precision. The picture changes for dimensions larger than 80, where VkD-CMA has worse ERT values on *every* problem. In dimension 160 VkD-CMA is about 2–4 times slower than LMCMA for all targets. In dimensions 320 and 640, VkD-CMA does not reach the most difficult targets anymore.

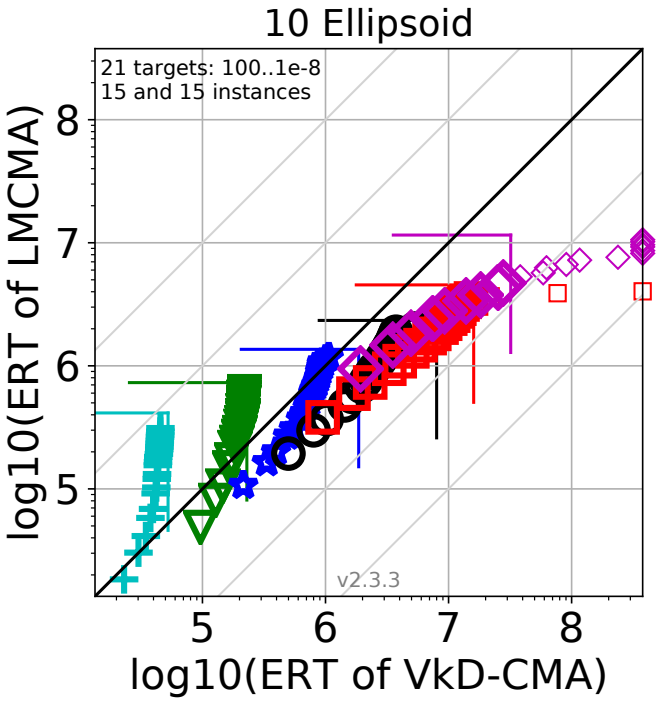

Figure 4.7: Expected running time (ERT in $\log_{10}$ of number of function evaluations) of LMCMA (y-axis) versus VkD-CMA (x-axis) for 21 target values between 10^{2} and 10^{-8} in each dimension on the Ellipsoid function. Colored markers represent dimension 20:+, 40:▽, 80:⋆, 160:○, 320:□, 640:◇. The rectangle indicates the maximal budget. Small markers indicate that values are computed from simulated restarts (due to some trials being unsuccessful) and markers on the figure edge indicate that the target was never reached by the respective algorithm.

Table 4.4: Excerpt of runtime (ERT) tables generated from COCO here in dimension 160

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f13								
LMCMA	**6418**(333)$^{\star 3}$	**1.8e5**(1e5)	9.7e6(1e7)	**1.1e8**(1e8)	∞	∞	∞ *8e6*	0/15
VkD-CMA	7454(904)	1.9e5(4e5)	**8.8e6**(9e6)	∞	∞	∞	∞ *8e6*	0/15
f14								
LMCMA	**1302**(117)	**3100**(263)$^{\star 3}$	**4439**(268)$^{\star 3}$	**6595**(324)$^{\star 4}$	**1.3e4**(679)$^{\star 4}$	**1.1e5**(6752)$^{\star 4}$	**1.6e6**(1e5)	14/15
VkD-CMA	1457(188)	3607(318)	5261(526)	8789(482)	1.9e4(2054)	2.0e5(4e4)	3.6e6(3e6)	0/15

Runtime (ERT) tables

Tables with the expected runtime to reach several target function values are also produced, for every function and dimension. Similarly to the scatter plots, they maintain information on single problems separately, but for a smaller set of target values. They are produced for data of any number of solvers. As an example, a part of the tables comparing LMCMA and VkD-CMA that contains information only for two test functions in dimension 160 is given in Table 4.4. In braces, the half difference between 10 and 90 percentiles of runtimes is shown as dispersion measure. The last column gives the number of successful trials to reach the most difficult target $\Delta f + 10^{-8}$. If this target is never reached, the median of conducted function evaluations is given in italics. Finally, a star indicates statistically significantly better results (according to the rank sum test) of a solver when compared to every other algorithm of the table, with $p = 0.05$ or $p = 10^{-k}$ where k is given after the star, and with Bonferroni correction with the number of functions (24).

4.2 Benchmark studies

We include in this section two benchmark studies, the first containing the comparison of large scale solvers on the `bbob-largescale` suite and the second compares the performance of various solvers from the `SciPy` optimization toolbox, on the `bbob` suite of COCO. They are independent of the main body of the thesis and can be viewed as supplementary contributions.

4.2.1 Benchmarking Variants of CMA-ES and L-BFGS-B on the `bbob-largescale` Testbed[9]

We present hereby a benchmarking comparison of large scale CMA-ES variants and of L-BFGS-B, as in section 2.3, and part of the shown results below coincide with those included in 2.3. In this subsection, though, instead of describing the algorithmic insights we focus more on the experimental part of the benchmarking process (e.g. CPU timing) as well as on gathering more results and under additional parameter settings of the considered solvers.

Parameter Setting of Algorithms

As in section 2.3, we benchmark VkD-CMA-ES [5, 6] with fixed $k = 2$ (denoted V2D in the results) and adaptive k parameter (denoted VkD), the original LMCMA implementation [58], denoted LMCMA14 (14lmcma in Tables 4.6 and 4.7), and a more recent version [60], LMCMA17 (17lmcma in Tables 4.6 and 4.7) under their default parameter setting, as well as the RmES algorithm [56], separable CMA-ES [81] and the quasi Newton L-BFGS-B algorithm.

In comparison to section 2.3, the following additions/modifications have been done: for the RmES algorithm, apart from considering $m = 2$ evolution paths (denoted R2ES) we also consider $m = 10$ (denoted R10ES). The predecessor of VkD-CMA-ES [5], i.e. the VD-CMA-ES algorithm [9] is hereby benchmarked. Also, for L-BFGS, we have considered for the parameter `maxcor` that controls the maximum number of variable metric corrections used for the Hessian approximation, apart from its default value (10), a setting where it is equal to $2 \times D$, D being the dimension, denoted m2DLBFGS in the results (we benchmark its implementation from the latest version[10] of the Python SciPy library). Separable CMA-ES is benchmarked in its Python implementation (available in the CMA-ES[11] implementation with the option of sampling from and adapting a distribution with a diagonal covariance matrix) and, finally, results of CMA-ES in its default setting, denoted CMA, are provided for problem dimensions up to 320.

Experimental Procedure

We run the algorithms on the entire `bbob-largescale` suite for $5 \times 10^4 D$ function evaluations according to [42]. A policy of independent restarts is followed when default termination conditions are met. Only for L-BFGS-B, the

[9] This subsection is based on a workshop paper [95] by K. Varelas, presented to the 2019 Genetic and Evolutionary Computation conference.
[10] Version 1.2.1
[11] `pycma`

parameter `ftol` that sets the f tolerance termination condition was changed to the machine precision[12] for very high accuracy. For all solvers, the initial point was chosen uniformly at random in $[-4, 4]^D$ and for CMA-ES and its variants, the initial step size was set to 2.

CPU Timing

The complete experiment was run on several multicore machines with different processor types and number of cores. In order to evaluate the CPU timing of each algorithm, we have performed a shorter experiment, running the solvers with restarts on the first 3 instances of each function of the `bbob-largescale` test suite for $100 \times D$ function evaluations. For this, we used (not exclusively) two Linux multicore machines. The time per function evaluation, measured in 10^{-5} seconds for dimensions 20, 40, 80, 160, 320, 640 along with the corresponding processor type and number of cores, are presented in Table 4.5. The MATLAB implementation of RmES was run with MATLAB R2019a.

Results

Results from experiments according to [42] and [41] on the benchmark functions given in [99] are presented in The experiments were performed with COCO [43, 44], version 2.2.1[13], the plots were produced with version 2.3.3.

The **average runtime (aRT)**, used in the figures and tables, depends on a given target function value, $f_{\mathrm{t}} = f_{\mathrm{opt}} + \Delta f$, and is computed over all relevant trials as the number of function evaluations executed during each trial while the best function value did not reach f_{t}, summed over all trials and divided by the number of trials that actually reached f_{t} [40, 79]. **Statistical significance** is tested with the rank-sum test for a given target Δf_{t} using, for each trial, either the number of needed function evaluations to reach Δf_{t} (inverted and multiplied by -1), or, if the target was not reached, the best Δf-value achieved, measured only up to the smallest number of overall function evaluations for any unsuccessful trial under consideration.

Observations

Additionally to the conclusions stated in section 2.3, we extract the following observations.

While the performance of LMCMA and RmES does not change between the original and rotated Ellipsoid function, as well as between the original and rotated Rosenbrock function, this is not the case for VkD-CMA. In particular, in dimension 320 the runtime is larger by at least a factor of 10 for the Ellipsoid function when rotations are applied, as presented in Figure 4.12. This function is an example that perfectly illustrates the tradeoff of a restricted covariance matrix model that exploits separability and of maintaining rotational invariance: with no rotation, sepCMA is the fastest method, more than 10 times faster than CMA for the most difficult targets. Then

[12] Equal to $2.220446049250313 \times 10^{-16}$

[13] The code that was used was under development, with no difference in the definition of the `bbob-largescale` testbed, which was officially included in version 2.3 of COCO

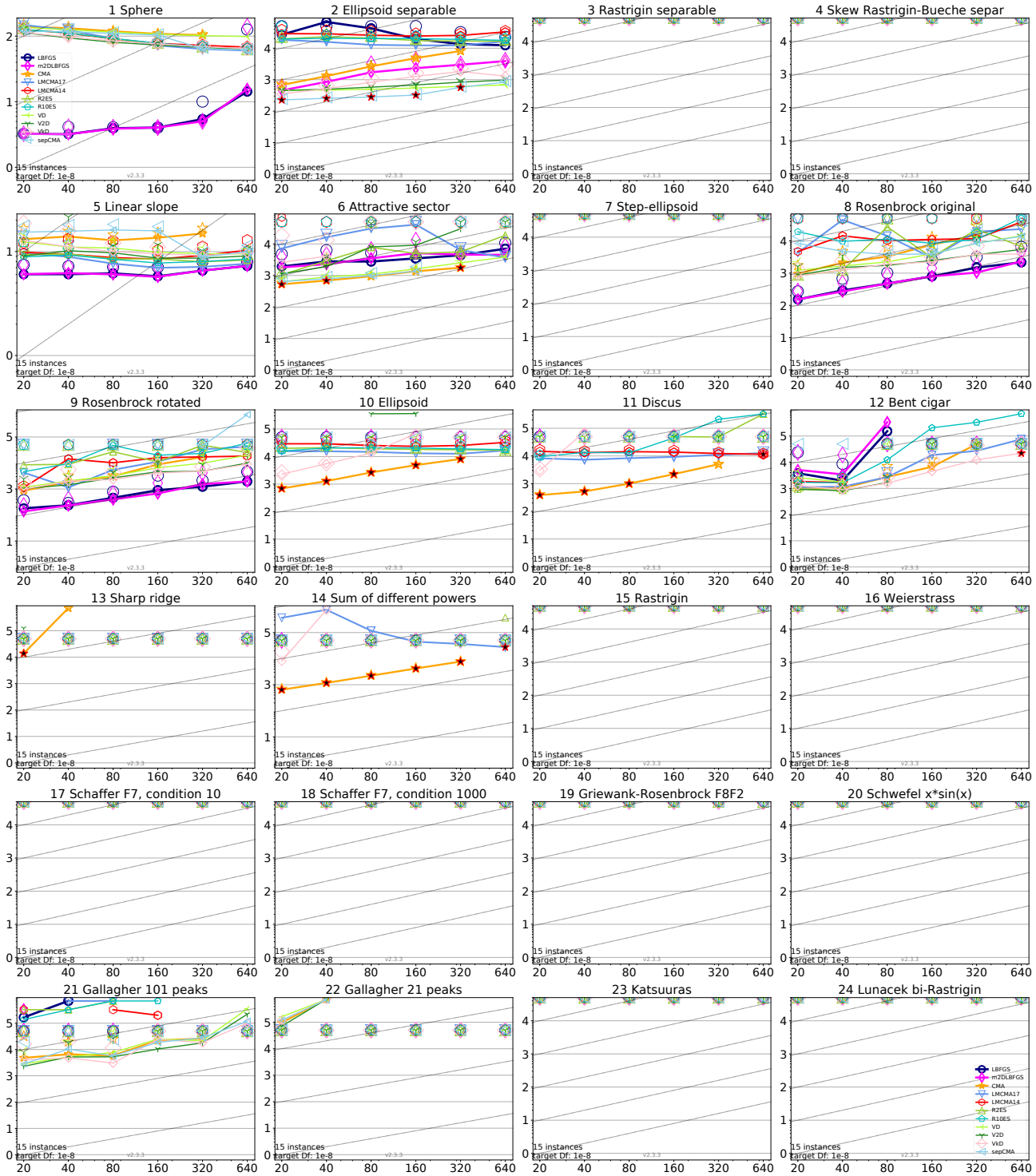

Figure 4.8: Average running time (aRT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: $\circ$: LMCMA, $\diamond$: VkD-CMA, $\star$: LMCMA14, $\triangledown$: LMCMA17, $\hexagon$: R10ES, $\triangle$: R2ES, $\pentagon$: V2D, $\prec$: VD, $\curlyvee$: VkD, $\diamond$: m2DLBFGS, $\triangleleft$: sepCMA.

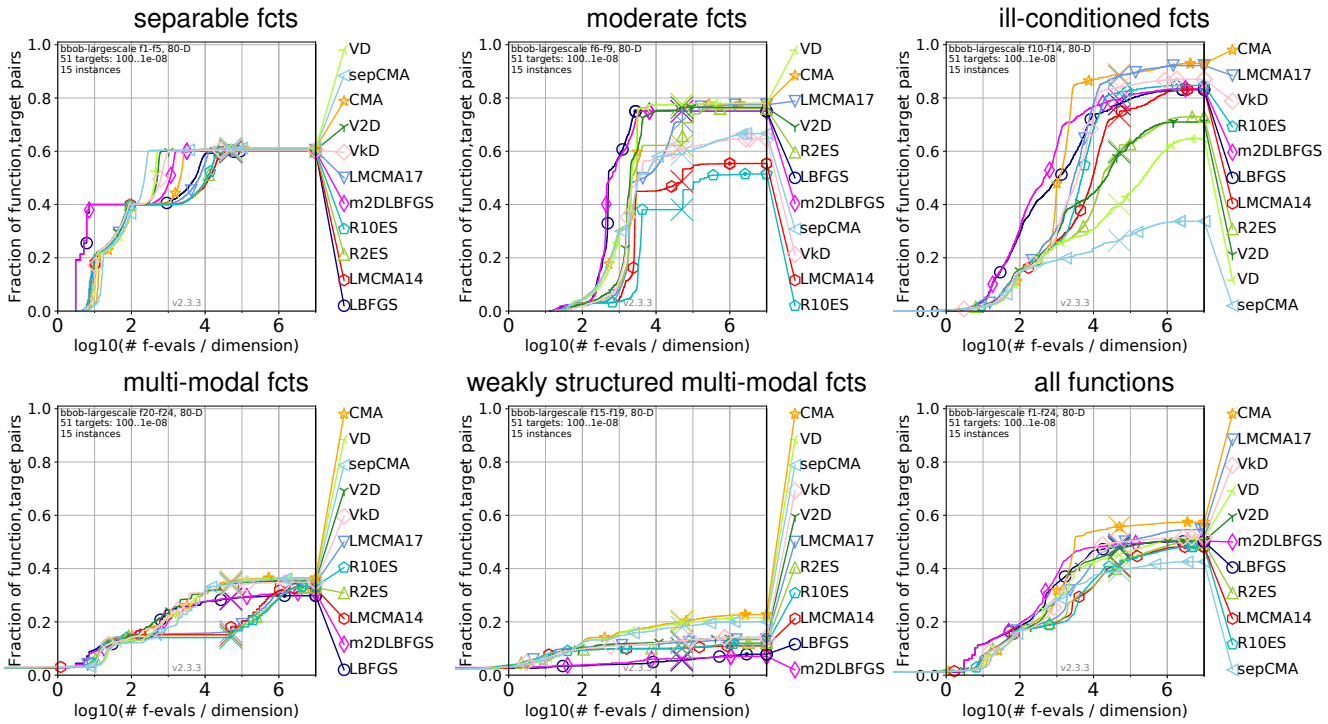

Figure 4.9: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 80-D.

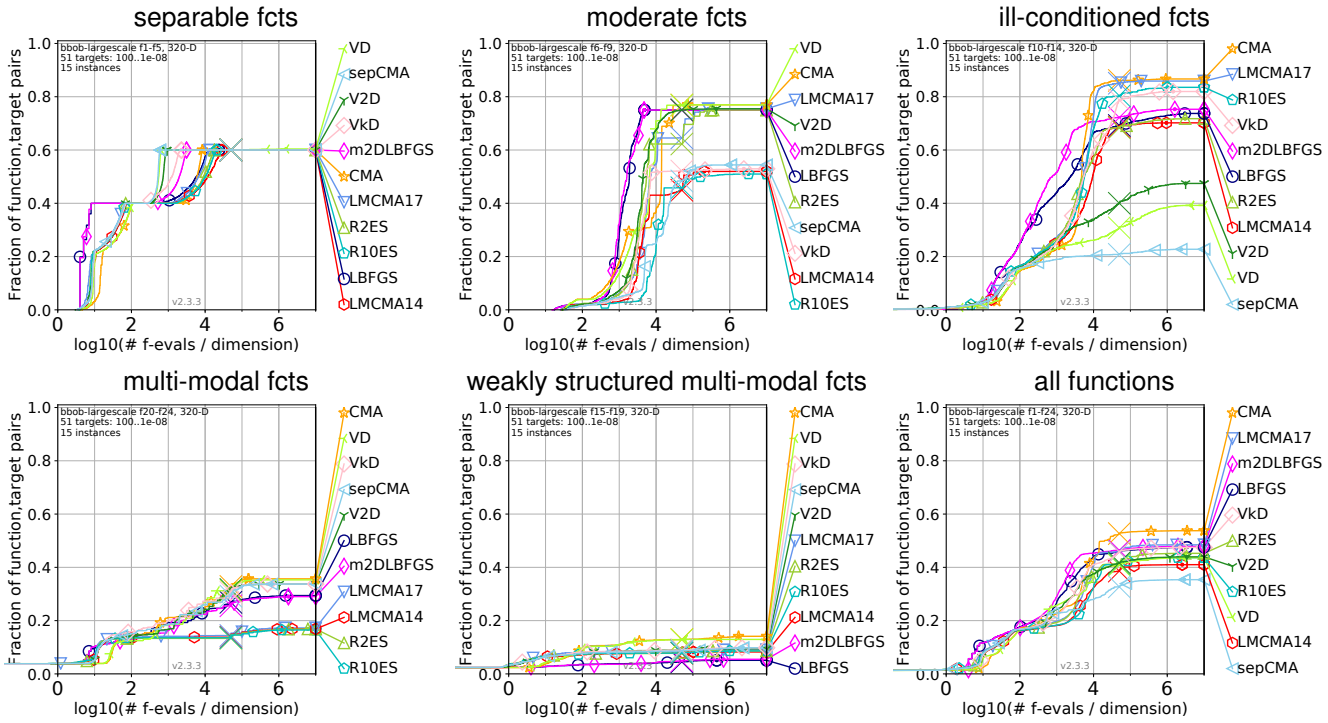

Figure 4.10: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 320-D.

Algorithm	Processor Type	20-D	40-D	80-D	160-D	320-D	640-D
sepCMA	Intel Core Haswell, no TSX @2.29 GHz-28 cores	19	23	34	56	100	200
LMCMA17	Intel Core Haswell, no TSX @2.29 GHz-28 cores	4.85	8.70	16.7	38.4	69.0	136
LMCMA14	Intel Core Haswell, no TSX @2.29 GHz-28 cores	5.54	9.40	18.7	38.2	77.9	149
R2ES	Intel(R) Xeon(R) CPU E5-2640 v4 @2.40GHz-40 cores	10.2	13.4	22.2	40.0	79.3	157.4
R10ES	Intel(R) Xeon(R) CPU E5-2640 v4 @2.40GHz-40 cores	9.4	13.2	22.5	39.1	79.8	156.7
VD-CMA	Intel Core Haswell, no TSX @2.29 GHz-28 cores	23	27	37	57	100	200
VkD-CMA	Intel Core Haswell, no TSX @2.29 GHz-28 cores	30	36	45	64	110	200
V2D-CMA	Intel Core Haswell, no TSX @2.29 GHz-28 cores	27	33	43	62	110	200
L-BFGS-B	Intel Core Haswell, no TSX @2.29 GHz-28 cores	2.9	5.1	9.3	18	35	70
m2DLBFGS	Intel Core Haswell, no TSX @2.29 GHz-28 cores	3.7	5.4	9.8	20	36	68
CMA	Intel Core Haswell, no TSX @2.29 GHz-28 cores	17	20	30	49	95	200

Table 4.5: CPU timing per function evaluation

VD with one principal search direction, V2D and VkD follow. When rotations are applied, the opposite effect appears, with sepCMA not reaching any target, and VD and V2D showing the worst performance right after sepCMA. The property of invariance under affine transformations of the search space that CMA-ES possesses does not hold for the restricted complexity model methods.

Looking at the aggregated ECDFs of all functions in Figures 4.9 and 4.10 in order to compare the number of evolution paths of RmES, the picture is diverse in dimension 80 but more clear in dimension 320, where R2ES dominates R10ES for all budgets. The latter can be of advantage though for certain functions. For the convex quadratic Discus function, in dimensions smaller or equal to 160, R10ES is preferable and in dimension 320 R2ES overtakes R10ES, while the picture is opposite for the Bent Cigar function where R2ES dominates R10ES only in dimension 20. This fact suggests that the parameter strongly relates to the number of short axes and a larger value provides more robustness for these functions. As a result, R10ES is clearly superior to R2ES on the group of ill-conditioned functions in dimension 80 and the performance difference becomes less significant in dimension 320.

Another observation is the small success rate of LMCMA and RmES for the Gallagher function `f21` in $320D$,

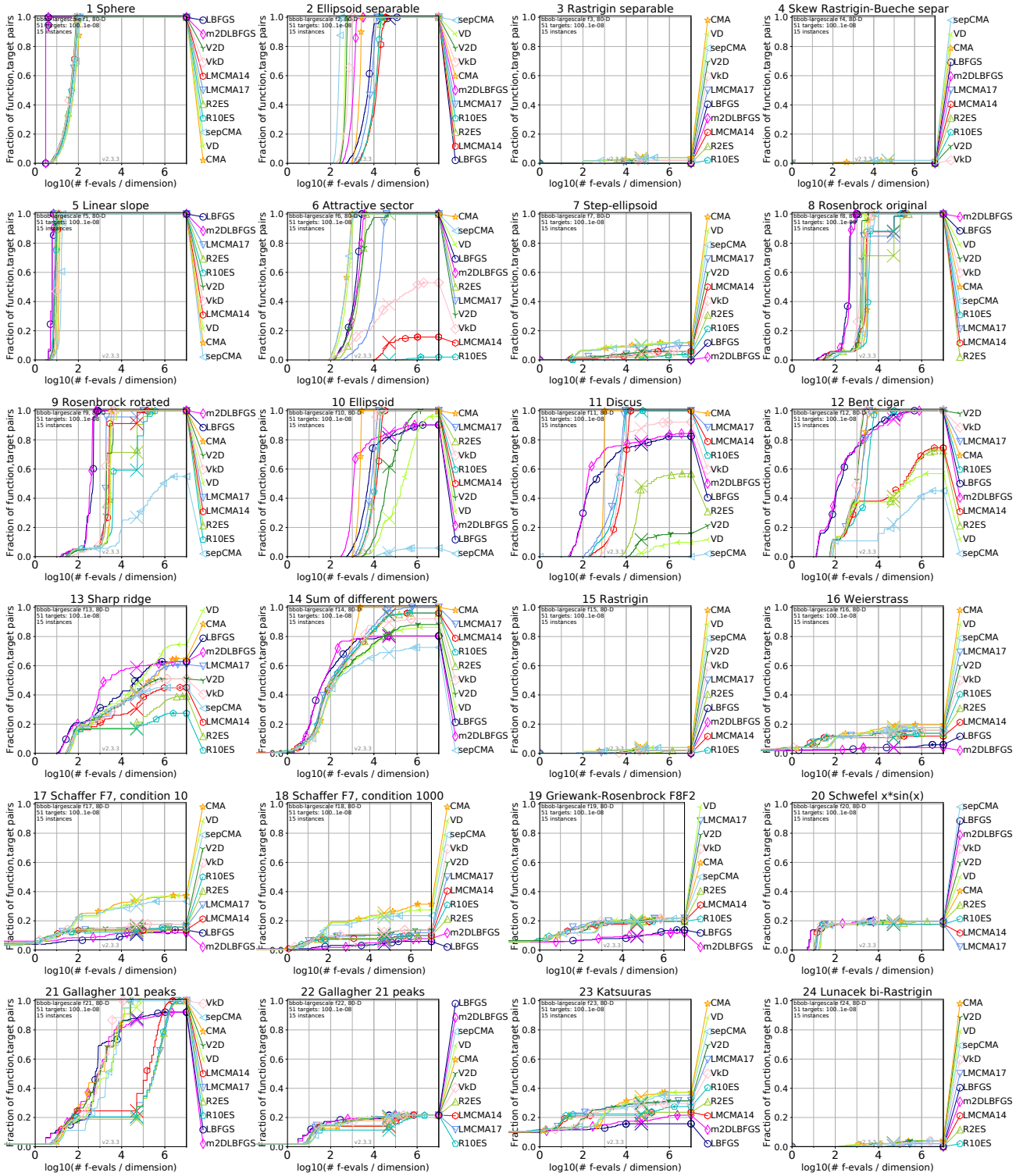

Figure 4.11: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 80.

illustrated in Figure 4.12. This is due to poor termination conditions of the specific implementations that employ only step size values compared to the other benchmarked solvers, for which the high success rate is attributed to the restart policy.

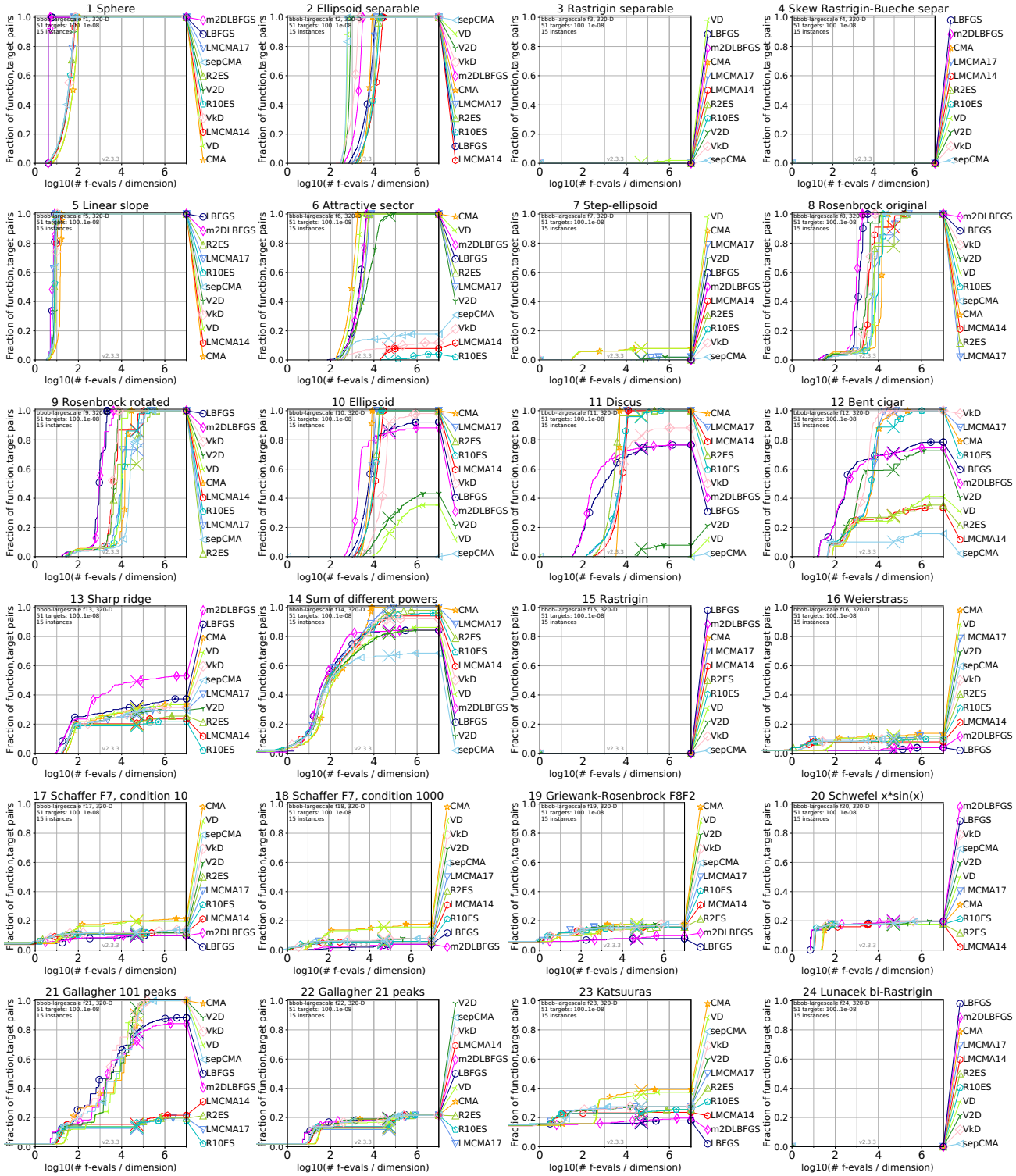

Figure 4.12: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 320.

Lastly, in the case of L-BFGS-B, increasing the maximum number of corrections is clearly of advantage, that affects mostly the group of ill-conditioned functions. Considering the example of the separable Ellipsoid function in dimension 320 depicted in Figure 4.12, the runtime is smaller by a factor of 2 for the easiest targets

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1								
LMCMA	**485**(0)	**485**(0)	**485**(0)	**485**(0)	**485**(0)	**485**(0)	**614**(80)	15/15
VkD-CMA	**485**(0)	**485**(0)	**485**(0)	**485**(0)	**485**(0)	**485**(0)	**614**(80)	15/15
CMA	2697(111)	4364(146)	5961(102)	7631(108)	9224(130)	1.2e4(192)	1.6e4(182)	15/15
17lmcma	1865(49)	2950(85)	4006(147)	5081(155)	6113(150)	8256(158)	1.0e4(132)	15/15
14lmcma	1946(109)	3071(58)	4220(170)	5384(155)	6513(207)	8877(185)	1.1e4(156)	15/15
R2ES	1774(84)	2960(164)	4082(134)	5199(161)	6318(209)	8732(161)	1.1e4(243)	15/15
R10ES	1886(150)	3053(166)	4294(154)	5391(143)	6627(160)	8961(249)	1.1e4(304)	15/15
VD	2676(136)	4265(284)	5860(158)	7504(168)	9114(188)	1.2e4(232)	1.5e4(303)	15/15
V2D	1728(156)	2797(237)	3895(215)	5018(233)	6099(359)	8290(272)	1.1e4(388)	15/15
VkD	1739(95)	2807(122)	3891(212)	5041(264)	6172(369)	8508(296)	1.1e4(517)	15/15
sepCMA	2577(50)	4157(110)	5703(116)	7280(160)	8820(125)	1.2e4(146)	1.5e4(238)	15/15
f2								
LMCMA	2.5e5(2e4)	4.0e5(2e4)	5.8e5(7e4)	7.4e5(1e5)	9.2e5(1e5)	1.3e6(1e5)	1.6e6(4e5)	13/15
VkD-CMA	1.2e5(4830)	1.5e5(7004)	1.9e5(7245)	2.1e5(8936)	2.5e5(1e4)	2.9e5(1e4)	3.3e5(8251)	15/15
CMA	3.7e5(2e4)	4.4e5(2e4)	5.1e5(1e4)	5.6e5(2e4)	6.1e5(2e4)	6.8e5(1e4)	7.5e5(3e4)	15/15
17lmcma	3.5e5(4e4)	5.3e5(5e4)	7.1e5(5e4)	8.8e5(1e5)	1.1e6(8e4)	1.4e6(1e5)	1.8e6(3e5)	15/15
14lmcma	5.3e5(6e4)	8.6e5(1e5)	1.2e6(2e5)	1.5e6(4e5)	1.9e6(3e5)	2.7e6(4e5)	3.5e6(4e5)	15/15
R2ES	5.9e5(6e4)	8.3e5(7e4)	1.1e6(6e4)	1.3e6(7e4)	1.6e6(9e4)	2.1e6(1e5)	2.6e6(1e5)	15/15
R10ES	5.5e5(4e4)	8.6e5(5e4)	1.2e6(9e4)	1.4e6(1e5)	1.7e6(8e4)	2.3e6(2e5)	2.9e6(3e5)	15/15
VD	**6.1e4**(4112)	**6.8e4**(5968)	**7.2e4**(5928)	**7.5e4**(3457)	**7.8e4**(3953)	**8.2e4**(4758)	**8.5e4**(4732)	15/15
V2D	6.5e4(3328)	7.6e4(1213)	8.5e4(4990)	9.1e4(5302)	9.5e4(4733)	1.0e5(4184)	1.1e5(2758)	15/15
VkD	6.9e4(2e4)	9.9e4(3e4)	1.2e5(1e4)	1.4e5(2e4)	1.5e5(2e4)	1.8e5(3e4)	1.9e5(5e4)	15/15
sepCMA	**3.3e4**(2902)*4	**3.7e4**(2168)*4	**3.9e4**(2958)*4	**4.2e4**(2338)*4	**4.4e4**(3288)*4	**4.7e4**(3049)*4	**5.0e4**(2160)*4	15/15
f3								
LMCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
CMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
17lmcma	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
14lmcma	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
R2ES	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
R10ES	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VD	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
V2D	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VkD	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
sepCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
f4								
LMCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
CMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
17lmcma	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
14lmcma	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
R2ES	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
R10ES	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VD	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
V2D	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VkD	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
sepCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
f5								
LMCMA	**764**(121)	**861**(80)	**925**(80)	**925**(80)	**925**(0)	**925**(80)	**925**(121)	15/15
VkD-CMA	**764**(121)	**861**(121)	**904**(121)	**914**(121)	**914**(80)	**914**(161)	**914**(80)	15/15
CMA	1648(148)	2019(216)	2157(328)	2171(234)	2172(211)	2172(165)	2172(164)	15/15
17lmcma	977(146)	1071(116)	1108(178)	1113(192)	1116(126)	1116(164)	1116(160)	15/15
14lmcma	1181(150)	1310(192)	1348(214)	1353(138)	1354(200)	1354(188)	1354(237)	15/15
R2ES	1146(70)	1310(125)	1349(88)	1352(119)	1358(79)	1358(64)	1358(95)	15/15
R10ES	1090(122)	1196(139)	1236(138)	1240(175)	1240(128)	1240(128)	1240(144)	15/15
VD	1323(132)	1515(129)	1556(139)	1574(146)	1577(154)	1577(142)	1577(162)	15/15
V2D	1178(45)	1332(203)	1373(122)	1383(245)	1383(286)	1383(346)	1383(214)	15/15
VkD	1217(167)	1420(132)	1499(119)	1521(152)	1521(129)	1521(172)	1521(125)	15/15
sepCMA	1810(101)	2321(164)	2500(141)	2525(98)	2531(200)	2531(204)	2531(91)	15/15
f6								
LMCMA	8.2e4(2e4)	1.4e5(1e4)	1.9e5(2e4)	2.4e5(2e4)	2.9e5(2e4)	3.9e5(2e4)	5.0e5(3e4)	15/15
VkD-CMA	8.2e4(3e4)	1.5e5(4e4)	2.2e5(3e4)	2.9e5(3e4)	3.6e5(5e4)	5.0e5(6e4)	6.5e5(4e4)	15/15
CMA	**3.7e4**(2021)	**5.4e4**(4119)	**7.3e4**(2808)	**9.2e4**(5969)	**1.1e5**(5962)	**1.5e5**(8452)	**1.9e5**(9728)	15/15
17lmcma	4.5e5(8e5)	9.4e5(1e6)	1.5e6(1e6)	2.1e6(2e6)	2.6e6(2e6)	3.9e6(2e6)	5.3e6(3e6)	12/15
14lmcma	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
R2ES	1.3e5(1e4)	2.0e5(2e4)	2.7e5(8416)	3.4e5(2e4)	4.2e5(1e4)	5.7e5(2e4)	7.2e5(8872)	15/15
R10ES	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VD	4.7e4(5410)	6.8e4(4153)	9.0e4(8778)	1.1e5(1e4)	1.4e5(1e4)	1.8e5(2e4)	2.4e5(2e4)	15/15
V2D	9.1e4(1e4)	1.7e5(3e4)	2.6e5(4e4)	3.8e5(2e5)	5.2e5(1e5)	8.2e5(1e5)	1.2e6(2e5)	15/15
VkD	2.6e6(2e6)	1.2e8(1e8)	∞	∞	∞	∞	*∞ 8e6*	0/15
sepCMA	**3.4e4**(3431)	**5.0e4**(5862)	**6.8e4**(4479)	**8.6e4**(1e4)	**1.1e5**(1e4)	**1.5e5**(2e4)	**2.0e5**(3e4)	15/15

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f7								
LMCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
CMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
17lmcma	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
14lmcma	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
R2ES	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
R10ES	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VD	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
V2D	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VkD	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
sepCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
f8								
LMCMA	**7.8e4**(2e4)	**9.6e4**(5e4)	**1.2e5**(2e5)	**1.2e5**(2e5)	**1.2e5**(2e5)	**1.2e5**(6e4)	**1.3e5**(2e5)	**1.3e5**(5e4)	15/15
VkD-CMA	**8.5e4**(2e4)	**1.1e5**(2e4)	**1.2e5**(5e4)	**1.2e5**(6e4)	**1.3e5**(3e4)	**1.3e5**(7e4)	**1.3e5**(6e4)	15/15
CMA	7.3e5(2e4)	9.4e5(2e5)	1.2e6(5e5)	1.2e6(5e5)	1.2e6(5e5)	1.2e6(3e5)	1.2e6(6e5)	15/15
17lmcma	3.4e5(3e4)	4.3e5(1e5)	4.4e5(1e5)	4.5e5(2e5)	4.6e5(1e5)	4.7e5(1e5)	4.7e5(1e5)	15/15
14lmcma	3.6e5(9e4)	4.9e5(2e5)	1.7e6(4e6)	1.8e6(1e5)	1.8e6(2e5)	1.8e6(4e6)	1.8e6(2e5)	13/15
R2ES	3.5e5(1e5)	4.6e5(8e4)	4.7e5(1e5)	4.8e5(1e5)	4.8e5(1e5)	4.8e5(2e5)	4.9e5(2e5)	15/15
R10ES	5.6e5(3e5)	7.0e5(3e5)	1.3e6(6e6)	1.3e6(4e5)	1.3e6(2e6)	1.3e6(2e6)	1.3e6(2e6)	14/15
VD	4.2e5(9662)	5.5e5(1e4)	6.0e5(9102)	6.1e5(8947)	6.2e5(3e5)	6.2e5(2e5)	6.3e5(1e4)	15/15
V2D	2.5e5(5e4)	3.1e5(5e4)	3.7e5(2e5)	3.7e5(8e4)	3.8e5(8e4)	3.9e5(7e4)	3.9e5(2e5)	15/15
VkD	2.5e5(5e4)	3.1e5(9e4)	3.3e5(1e5)	3.4e5(1e5)	3.4e5(1e5)	3.5e5(1e5)	3.5e5(8e4)	15/15
sepCMA	3.1e5(1e4)	4.1e5(2e4)	5.3e5(3e5)	5.4e5(3e5)	5.5e5(1e5)	5.8e5(3e5)	6.1e5(1e5)	15/15
f9								
LMCMA	**7.2e4**(3e4)	**8.5e4**(3e4)	**1.4e5**(2e5)	**1.4e5**(4e4)	**1.4e5**(2e5)	**1.4e5**(2e5)	**1.4e5**(2e4)	15/15
VkD-CMA	**7.9e4**(2e4)	**9.4e4**(3e4)	**1.0e5**(1e5)	**1.1e5**(6e4)	**1.1e5**(3e4)	**1.1e5**(1e5)	**1.1e5**(6e4)	15/15
CMA	7.4e5(1e4)	1.0e6(3e4)	1.4e6(5e5)	1.4e6(3e5)	1.4e6(5e5)	1.4e6(3e5)	1.4e6(5e5)	15/15
17lmcma	3.9e5(2e4)	5.3e5(6e4)	1.8e6(2e6)	1.8e6(1e5)	1.8e6(6e6)	1.8e6(6e6)	1.8e6(4e6)	13/15
14lmcma	4.1e5(8e4)	5.4e5(1e5)	2.5e6(8e6)	2.5e6(4e6)	2.5e6(4e6)	2.5e6(6e6)	2.6e6(6e6)	12/15
R2ES	3.6e5(5e4)	4.8e5(7940)	1.7e6(4e6)	1.7e6(4e6)	1.8e6(2e6)	1.8e6(4e6)	1.8e6(4e6)	13/15
R10ES	8.9e5(2e4)	1.1e6(2e5)	3.2e6(6e6)	3.2e6(6e6)	3.2e6(2e6)	3.2e6(4e6)	3.2e6(6e6)	12/15
VD	5.9e5(3e4)	7.8e5(1e5)	1.0e6(9e5)	1.0e6(9e5)	1.0e6(4e5)	1.0e6(2e5)	1.0e6(4e5)	15/15
V2D	3.6e5(5e4)	4.9e5(8e4)	7.1e5(5e5)	7.2e5(5e5)	7.2e5(3e5)	7.3e5(3e5)	7.3e5(3e5)	15/15
VkD	3.6e5(9e4)	4.6e5(8e4)	6.4e5(3e5)	6.4e5(3e5)	6.5e5(2e5)	6.6e5(3e5)	6.6e5(4e5)	15/15
sepCMA	2.0e6(9e4)	1.1e8(1e8)	1.1e8(1e8)	1.1e8(2e8)	1.1e8(1e8)	∞	*∞ 8e6*	0/15
f10								
LMCMA	**2.5e5**(3e4)	**3.9e5**(4e4)	5.6e5(6e4)	7.2e5(1e5)	9.1e5(9e4)	1.3e6(2e5)	*∞ 8e6*	0/15
VkD-CMA	**1.2e5**(5072)*4	**1.6e5**(7084)*4	**1.9e5**(7849)*4	**2.1e5**(6198)*4	**2.4e5**(5554)*4	**3.9e5**(1e5)*4	1.1e8(1e8)	0/15
CMA	3.7e5(2e4)	4.4e5(4e4)	**5.0e5**(1e4)	**5.6e5**(9718)	**6.0e5**(2e4)	**6.8e5**(2e4)	**7.5e5**(2e4)*2	15/15
17lmcma	3.7e5(4e4)	5.6e5(3e4)	7.6e5(8e4)	9.5e5(1e5)	1.1e6(1e5)	1.5e6(2e5)	**1.9e6**(2e5)	15/15
14lmcma	5.1e5(7e4)	8.6e5(1e5)	1.2e6(2e5)	1.5e6(2e5)	1.9e6(3e5)	2.6e6(3e5)	3.4e6(5e5)	15/15
R2ES	5.8e5(7e4)	8.4e5(7e4)	1.1e6(1e5)	1.4e6(1e5)	1.6e6(2e5)	2.1e6(3e5)	2.7e6(2e5)	15/15
R10ES	5.3e5(3e4)	8.1e5(9e4)	1.1e6(2e5)	1.4e6(3e5)	1.7e6(2e5)	2.3e6(3e5)	3.0e6(3e5)	15/15
VD	3.3e6(2e6)	5.5e6(7e6)	1.1e7(8e6)	2.8e7(5e7)	1.2e8(2e8)	∞	*∞ 8e6*	0/15
V2D	1.6e6(7e5)	3.1e6(2e6)	4.8e6(3e6)	7.4e6(3e6)	1.7e7(1e7)	5.6e7(7e7)	5.7e7(6e7)	2/15
VkD	1.0e6(4e5)	1.6e6(3e5)	2.1e6(4e5)	2.5e6(5e5)	2.9e6(5e5)	3.4e6(6e5)	3.7e6(3e5)	9/15
sepCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
f11								
LMCMA	**1.1e4**(2777)	**1.7e4**(3582)	**2.4e4**(5152)	**3.5e4**(2e4)	**1.1e5**(9e4)	2.0e6(2e6)	*∞ 8e6*	0/15
VkD-CMA	**1.1e4**(3381)	**1.7e4**(2858)	**2.3e4**(5716)	**2.7e4**(6561)	**4.1e4**(3e4)*2	**4.1e5**(7e5)	*∞ 8e6*	0/15
CMA	2.7e5(5103)	2.9e5(6805)	3.0e5(6118)	3.1e5(3660)	3.2e5(4136)	**3.4e5**(3404)	**3.5e5**(2715)*4	15/15
17lmcma	7.7e4(7665)	1.9e5(1e4)	3.5e5(3e4)	5.1e5(2e4)	6.8e5(3e4)	1.0e6(3e4)	**1.3e6**(4e4)	15/15
14lmcma	2.5e5(6e4)	4.6e5(5e4)	6.6e5(3e4)	9.0e5(5e4)	1.1e6(9e4)	1.5e6(1e5)	2.0e6(1e5)	15/15
R2ES	6.8e5(1e5)	1.1e6(1e5)	1.4e6(2e5)	1.8e6(1e5)	2.2e6(2e5)	2.9e6(2e5)	3.6e6(3e5)	10/15
R10ES	1.4e5(4e4)	3.4e5(7e4)	5.4e5(9e4)	7.4e5(1e5)	9.4e5(1e5)	1.3e6(2e5)	1.7e6(2e5)	9/15
VD	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
V2D	3.7e7(4e7)	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
VkD	2.7e5(7e4)	4.9e5(2e5)	7.0e5(2e5)	8.9e5(4e5)	1.1e6(5e5)	1.8e6(3e5)	1.1e8(2e8)	0/15
sepCMA	∞	∞	∞	∞	∞	∞	*∞ 8e6*	0/15
f12								
LMCMA	**4446**(3663)	**1.3e4**(8292)	**1.9e4**(1e4)	**4.9e4**(4e4)	**7.2e4**(4e4)	1.2e6(3e6)	*∞ 8e6*	0/15
VkD-CMA	**3555**(1167)	**1.1e4**(6078)	**2.2e4**(3e4)	**3.8e4**(4e4)	**7.5e4**(6e4)	1.0e6(2e6)	*∞ 8e6*	0/15
CMA	2.9e4(753)	8.1e4(4e4)	2.1e5(1e5)	3.3e5(9e4)	4.2e5(8e4)	**6.6e5**(1e5)	**8.8e5**(3e5)	15/15
17lmcma	2.5e4(2e4)	7.7e4(7e4)	1.9e5(1e5)	3.2e5(2e5)	4.4e5(1e5)	6.9e5(3e5)	1.4e6(7e5)	14/15
14lmcma	2.2e4(2e4)	7.7e4(2e4)	1.8e6(2e6)	8.1e6(8e6)	1.2e8(1e8)	∞	*∞ 8e6*	0/15
R2ES	2.1e4(2e4)	1.1e6(2e6)	9.9e6(1e7)	3.2e7(2e7)	∞	∞	*∞ 8e6*	0/15
R10ES	2.4e4(3e4)	1.1e5(1e5)	2.9e5(3e5)	4.8e5(3e5)	5.8e5(2e5)	7.6e5(2e5)	3.8e6(6e6)	3/15
VD	3.1e4(2e4)	4.7e5(2e4)	1.0e7(2e7)	1.2e8(1e8)	∞	∞	*∞ 8e6*	0/15
V2D	2.4e4(9682)	6.4e4(1e4)	1.3e5(5e4)	1.7e5(7e4)	7.9e5(4e6)	2.2e7(1e7)	*∞ 8e6*	0/15
VkD	3.0e4(1e4)	7.1e4(2e4)	1.4e5(7e4)	2.0e5(6e4)	2.9e5(2e5)	**4.2e5**(3e5)	**5.9e5**(2e5)*	15/15
sepCMA	2.5e6(5e6)	∞	∞	∞	∞	∞	*∞ 8e6*	0/15

Table 4.6: Average runtime (aRT) to reach given targets, measured in number of function evaluations, in dimension 160 for the first 12 functions. For each function, the aRT and, in braces as dispersion measure, the half difference between 10 and 90%-tile of (bootstrapped) runtimes is shown for the different target Δf-values as shown in the top row. #succ is the number of trials that reached the last target $f_{\mathrm{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p = 0.05$ or $p = 10^{-k}$ when the number k following the star is larger than 1, with Bonferroni correction by the number of functions (24). Best results are printed in bold.

and this factor increases up to 6 for the most difficult targets. This configuration can imply a slight defect, e.g. for the Bent Cigar function, but in overview it dominates the default setting as can be seen from the aggregated ECDFs of Figures 4.9 and 4.10, where for all budgets the success rate is superior.

4.2.2 Benchmarking Multivariate Solvers of SciPy on the Noiseless Testbed[14]

A variety of solvers either in a black-box setting or in a gradient based setting with approximation of the gradients is benchmarked on the `bbob` testbed [38] in this subsection. In particular, multivariate optimization solvers from the latest version of the Python SciPy[15] library are compared, under default or modified parameter set-

[14]This subsection is based on a workshop paper [96] by K. Varelas and M. A. Dahito, presented to the 2019 "Genetic and Evolutionary Computation conference".

[15]Version: 1.2.1

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f13								
LMCMA	**3920**(201)	**4.1e4**(5e4)	**2.5e6**(3e6)	**5.7e7**(4e7)	∞	∞	∞ *8e6*	0/15
VkD-CMA	**3952**(282)	**4.0e4**(8895)	**1.9e5**(4e5)^{*2}	**3.3e5**(3e5)^{*2}	**4.6e6**(1e7)^{*4}	∞	∞ *8e6*	0/15
CMA	9761(362)	6.0e4(7e4)	6.2e6(4e6)	∞	∞	∞	∞ *8e6*	0/15
17lmcma	6418(251)	1.8e5(2e5)	9.7e6(1e7)	1.1e8(1e8)	∞	∞	∞ *8e6*	0/15
14lmcma	7388(456)	7.3e6(8e6)	3.5e7(4e7)	∞	∞	∞	∞ *8e6*	0/15
R2ES	6910(527)	3.2e7(2e7)	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	7512(804)	1.1e8(7e7)	∞	∞	∞	∞	∞ *8e6*	0/15
VD	9448(360)	1.1e5(1e5)	4.0e6(5e6)	∞	∞	∞	∞ *8e6*	0/15
V2D	7253(651)	2.4e5(4e5)	7.8e6(8e6)	∞	∞	∞	∞ *8e6*	0/15
VkD	7454(709)	1.9e5(4e5)	8.8e6(1e7)	∞	∞	∞	∞ *8e6*	0/15
sepCMA	8788(410)	8.7e4(1e5)	6.5e6(5e6)	∞	∞	∞	∞ *8e6*	0/15
f14								
LMCMA	**914**(161)	**1913**(282)	**2793**(523)	**4231**(564)	**7730**(402)	**7.9e4**(7245)	∞ *8e6*	0/15
VkD-CMA	**957**(161)	**2063**(322)	**2771**(402)	**4274**(322)	**8063**(483)	**3.5e4**(604)^{*4}	∞ *8e6*	0/15
CMA	2407(215)	5163(342)	7698(171)	1.3e4(538)	3.0e4(1115)	1.6e5(7500)	**4.7e5**(1e4)^{*4}	15/15
17lmcma	1302(126)	3100(267)	4439(250)	6595(383)	1.3e4(805)	1.1e5(4497)	**1.6e6**(8e4)	14/15
14lmcma	1418(119)	3507(164)	5133(186)	7982(198)	1.5e4(856)	1.4e5(6753)	3.2e6(2e5)	0/15
R2ES	1338(133)	3267(150)	4819(135)	7442(429)	1.6e4(519)	1.6e5(1e4)	2.5e6(1e5)	0/15
R10ES	1280(121)	3440(291)	5069(323)	8172(320)	1.7e4(1045)	1.5e5(1e4)	3.5e6(4e5)	0/15
VD	2238(395)	4940(331)	7537(321)	1.3e4(437)	2.6e4(1740)	3.6e5(1e5)	∞ *8e6*	0/15
V2D	1357(220)	3491(406)	5131(264)	8160(322)	1.8e4(904)	3.5e5(1e5)	∞ *8e6*	0/15
VkD	1457(138)	3607(428)	5261(407)	8789(172)	1.9e4(1849)	2.0e5(5e4)	3.6e6(4e6)	0/15
sepCMA	2264(268)	4854(375)	7247(464)	1.2e4(468)	2.4e4(2136)	1.5e6(5e5)	∞ *8e6*	0/15
f15								
LMCMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VD	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
f16								
LMCMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	**4.6e4**(2e5)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	9.7e4(9e4)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	1.1e8(2e8)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	2.2e7(2e7)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	5.2e7(6e7)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VD	**1.8e4**(3650)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	1.0e6(1e6)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	1.9e6(2e6)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	2.9e5(2e5)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
f17								
LMCMA	1.1e8(1e8)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	2.0e7(2e7)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	3126(502)	**8.2e4**(8e4)	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	**1509**(168)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	1.2e6(7778)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	1.2e6(2e6)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	3759(4721)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VD	2810(396)	**1.5e5**(2e5)	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	**1922**(733)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	2524(1932)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	3055(758)	6.4e5(8e5)	∞	∞	∞	∞	∞ *8e6*	0/15
f18								
LMCMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	**9040**(1326)	**1.2e8**(2e8)	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VD	**9293**(1863)	**1.1e8**(9e7)	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	9359(2106)	∞	∞	∞	∞	∞	∞ *8e6*	0/15

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f19								
LMCMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	4096(2157)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	**1251**(269)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	1426(239)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	**1330**(161)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	1359(206)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VD	3558(1365)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	4989(2233)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	3987(2615)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	4198(2338)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
f20								
LMCMA	**1226**(80)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	**1226**(80)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	4211(198)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	2085(154)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	2070(109)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	1850(82)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	2007(78)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VD	3793(359)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	1876(107)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	1988(200)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	3858(228)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
f21								
LMCMA	**3877**(6602)	1.1e6(2e6)	2.2e6(2e6)	2.2e6(4e6)	2.2e6(3e6)	2.2e6(4e6)	1.2e8(1e8)	0/15
VkD-CMA	**4038**(7367)	**4.9e5**(2e6)	**9.8e5**(2e5)	**9.8e5**(2e6)	**9.9e5**(2e5)	**9.9e5**(1e6)	∞ *8e6*	0/15
CMA	1.2e4(1e4)	1.2e6(4e6)	3.5e6(5e6)	3.5e6(6e6)	3.5e6(5e6)	3.5e6(5e6)	3.5e6(7e6)	12/15
17lmcma	5.3e6(8e6)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	2.0e6(4e6)	2.2e7(2e7)	3.2e7(4e7)	3.2e7(4e7)	3.2e7(3e7)	3.2e7(3e7)	3.2e7(4e7)	3/15
R2ES	1.2e6(1326)	1.1e8(7e7)	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	1.2e6(299)	5.2e7(6e7)	1.1e8(2e8)	1.1e8(7e7)	1.1e8(1e8)	1.1e8(1e8)	1.1e8(1e8)	1/15
VD	8017(2e4)	1.6e6(6e6)	3.9e6(8e6)	3.9e6(8e6)	4.0e6(6e6)	4.0e6(6e6)	4.0e6(1e7)	11/15
V2D	1.3e4(3e4)	**4.4e5**(5e5)	**1.7e6**(3e6)	**1.7e6**(4e6)	**1.7e6**(4e6)	**1.7e6**(2e6)	**1.7e6**(2e6)	14/15
VkD	1.3e4(1e4)	1.0e6(1e5)	3.9e6(6e6)	3.9e6(4e6)	3.9e6(8e6)	3.9e6(4e6)	3.9e6(4e6)	11/15
sepCMA	8201(3e4)	7.6e5(7e5)	3.0e6(2e6)	3.0e6(6e6)	3.0e6(5e6)	3.0e6(5e6)	**3.0e6**(3e6)	13/15
f22								
LMCMA	**2.1e4**(1e5)	5.9e6(1e7)	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	3.2e4(4026)	**5.7e6**(7e6)	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	8.3e4(2e5)	7.7e6(6e6)	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	4.0e6(8e6)	5.2e7(7e7)	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	5.3e6(8e6)	2.2e7(1e7)	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	5.3e6(8e6)	3.2e7(3e7)	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	2.0e6(2e6)	1.1e8(8e7)	∞	∞	∞	∞	∞ *8e6*	0/15
VD	6.0e4(8e4)	7.6e6(6e6)	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	1.5e5(5e5)	6.7e6(2e7)	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	**2.1e4**(6e4)	7.0e6(1e7)	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	3.6e4(6e4)	**5.4e6**(6e6)	∞	∞	∞	∞	∞ *8e6*	0/15
f23								
LMCMA	**1**(0)	∞	∞	∞^{*4}	∞^{*4}	∞^{*4}	∞ *8e6*^{*4}	0/15
VkD-CMA	**1**(0)	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	57(0)	1.0e5(8000)	**7.2e5**(5e5)	∞	∞	∞	∞ *8e6*	0/15
17lmcma	**1**(0)	**1622**(211)	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	**1**(0)	**5172**(1e4)	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	**1**(0)	2.9e6(8e6)	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	**1**(0)	2.0e6(2e6)	∞	∞	∞	∞	∞ *8e6*	0/15
VD	22(157)	1.1e5(4e4)	**2.7e6**(3e6)	∞	∞	∞	∞ *8e6*	0/15
V2D	**1**(0)	8922(5388)	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	6.3(0)	7862(3415)	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	**1**(0)	1.0e5(6e4)	4.0e6(4e6)	∞	∞	∞	∞ *8e6*	0/15
f24								
LMCMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD-CMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
CMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
17lmcma	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
14lmcma	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R2ES	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
R10ES	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VD	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
V2D	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
VkD	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15
sepCMA	∞	∞	∞	∞	∞	∞	∞ *8e6*	0/15

Table 4.7: Average runtime (aRT) to reach given targets, measured in number of function evaluations, in dimension 160 for the last 12 functions. For each function, the aRT and, in braces as dispersion measure, the half difference between 10 and 90%-tile of (bootstrapped) runtimes is shown for the different target Δf-values as shown in the top row. #succ is the number of trials that reached the last target $f_{\mathrm{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p = 0.05$ or $p = 10^{-k}$ when the number k following the star is larger than 1, with Bonferroni correction by the number of functions (24). Best results are printed in bold.
Data produced with COCO v2.3.2.5

tings. A similar study for a previous version of SciPy, that benchmarked six solvers of the library under default parameters has been presented in [15], where the Basin Hopping [100] restart strategy was used within each independent restart. It is useful to compare, though, how particular implementations of such methods are evolved and improved over time.

In this study we follow a policy where independent restarts are applied when the corresponding termination criteria are met, until a given budget of function evaluations is exhausted. Based on a preliminary experimentation, we choose proper parameter settings and termination conditions for some algorithms such that their performance is not deteriorated.

The contribution in comparison to [15] is threefold: For the common benchmarked solvers, we compare the different parameter settings and restart policies. Furthermore, complete data sets for all dimensions are included (in [15] the results were restricted to dimensions 2, 5 and 20) and three additional solvers are bench-

marked.

Benchmarked Algorithms and their Parameter Setting

In order to investigate such effects and identify proper settings prior to the performance comparison of all solvers, experimentation was performed separately up to some extent, concerning in most, but not all, cases the termination tolerances in search and objective space. The following algorithms were benchmarked, where a star indicates those included in [15] as described above: Nelder-Mead*, Powell*, BFGS*, L-BFGS-B*, Conjugate Gradient*, Truncated Newton, Differential Evolution, COBYLA and SLSQP*.

In the case of the quasi Newton L-BFGS-B algorithm [57] for high dimensional optimization, reducing the tolerance in objective values (`ftol` parameter with default value 10^{-8}) can be of advantage, in particular for ill-conditioned functions and for the Attractive Sector function, as presented in Figures 4.13 and 4.14. This performance improvement becomes more significant with increasing dimensionality. Thus, in our experimentation it was set to the float machine precision[16] for very high accuracy. More importantly, the maximum number of variable metric corrections for the Hessian approximation has to be set carefully. The default value is 10 and experiments showed performance improvement for increasing values up to $2 \times D$, D being the problem dimension, as illustrated in Figure 4.15. Thus it was set to this value in our comparison. Furthermore, the effect of decreasing the step length for the finite difference approximation of the gradient was investigated to some extent: decreasing the default value of this parameter (10^{-8}) can improve the performance on particular functions, such as the Ellipsoid, while it shows worse success ratio on others. A more detailed study was presented in [18] and in the following comparison it is set to its default value.

The Nelder-Mead [66] simplex method is tested both in its default setting and with adaptation of parameters to the dimensionality of the problem [32], controlled by the `adaptive` flag.

For the modified Powell's conjugate direction algorithm [76, 78], the parameter `ftol` was set to 10^{-15}. As for the termination tolerance in the search space, different values of `xtol` were tested in the set $\{10^{-2}, 10^{-3}, 10^{-5}, 10^{-6}\}$. Values larger than the default (10^{-4}) typically can make the solver faster only for the easiest targets, while smaller values can show an improved success rate for high budgets. In the following the default value was chosen.

The truncated Newton algorithm [68, 65] requires an estimation of the optimal f value. Since it always lies in $[-1000, 1000]$ [38] and in accordance to the black-box setting where no prior information is available for the function, we set this value to -1000.

The SLSQP method that uses Sequential Least-Squares Programming [51] has been tested for different values of `ftol` (10^{-6}, 10^{-9}, 10^{-12} and 10^{-15}). Same as L-BFGS-B and Powell's algorithm, it was sensitive to this parameter, that was set to 10^{-15} in the performance comparison.

The original BFGS [68] method, the conjugate gradient algorithm by Polak and Ribiere [68], the global optimization Differential Evolution [85] method as well as the Constrained Optimization BY Linear Approximation (COBYLA) algorithm [77] are benchmarked in their default setting.

[16] Equal to $2.220446049250313 \times 10^{-16}$

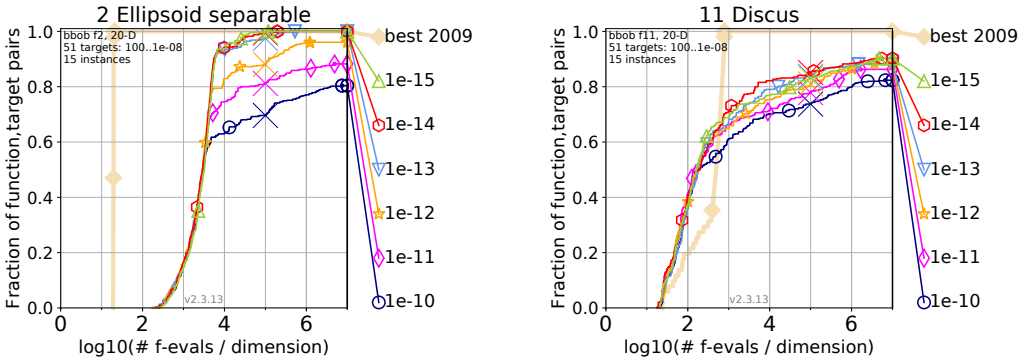

Figure 4.13: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ of the ill-conditioned separable Ellipsoid and Discus functions in 20-D for L-BFGS-B. The graphs correspond to different values of f-tolerance for termination.

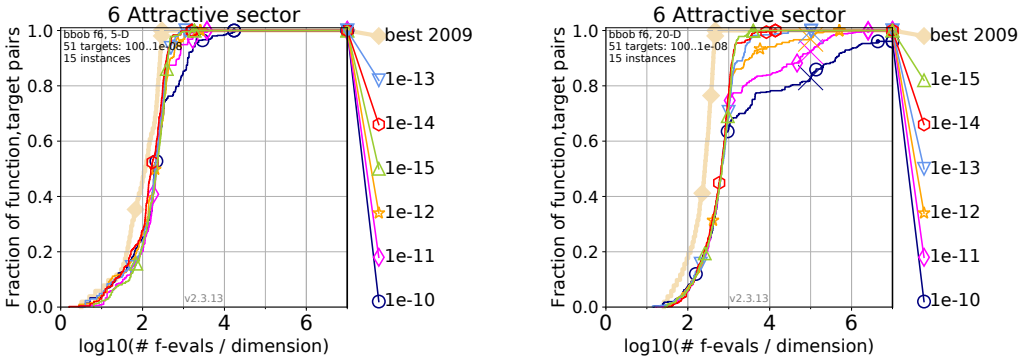

Figure 4.14: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ of the Attractive Sector function in 5-D and 20-D for L-BFGS-B. The graphs correspond to different values of f-tolerance for termination.

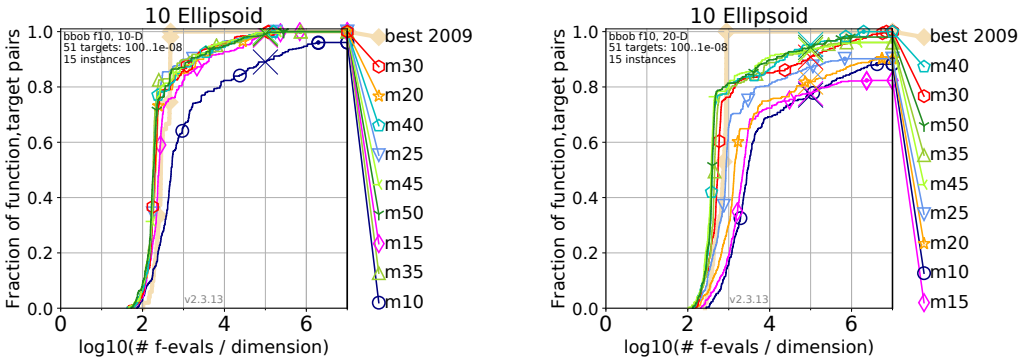

Figure 4.15: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ of the ill-conditioned separable Ellipsoid function in 10-D and 20-D for L-BFGS-B. The graphs correspond to different values of maximum number of variable metric corrections for the Hessian approximation.

In cases where the solver supported constraint handling, no constraints were applied. Finally, the maximum iterations were set to values large enough (wherever applicable), in order to avoid termination before convergence.

Algorithm	Processor Type	2-D	3-D	5-D	10-D	20-D	40-D
Nelder-Mead	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.2	2.3	2.4	2.9	3.5	5.0
Adaptive Nelder-Mead	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.1	2.1	2.2	2.6	3.3	4.8
Powell	Intel Core Haswell, no TSX @ 2.29GHz	2.1	2.5	2.4	2.5	2.9	4.0
BFGS	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.7	2.7	2.6	2.7	3.3	5.2
L-BFGS-B	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz	2.9	2.3	2.1	2.3	3.1	5.4
Conjugate Gradient	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.2	1.9	1.4	1.3	1.9	3.8
Truncated Newton	Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz	1.4	1.4	2.5	2.0	2.8	5.0
Differential Evolution	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz	8.4	8.4	8.5	9.3	11.0	14.0
COBYLA	Intel Core Haswell, no TSX @ 2.29GHz	0.51	0.53	0.65	0.96	2.1	8.2
SLSQP	Intel Core Haswell, no TSX @ 2.29GHz	2.0	1.9	1.8	2.1	1.9	2.2

Table 4.8: CPU timing per function evaluation

Experimental Procedure

All solvers were run on the `bbob` testbed with restarts for a maximum budget of $10^5 \times D$ function evaluations (at minimum, for solvers that did not support a termination callback such as COBYLA). For all runs, the initial point was chosen uniformly at random in $[-4, 4]^D$ and with the function value evaluated at this point. In the special case of Differential Evolution where no initial point is given, the domain bounds were set as $[-5, 5]^D$.

CPU Timing

The Python code was run on several multicore machines (not exclusively) with different number of cores. The time per function evaluation, measured in 10^{-5} seconds, for different dimensions along with the corresponding processor type is presented in Table 4.8.

Results

Results from experiments according to [42] and [41] on the benchmark functions given in [38] are presented in Figures 4.16, 4.17, 4.18, 4.19, and 4.20. The experiments were performed with COCO [43, 44], version 2.3,

the plots were produced with version 2.3. The solvers benchmarked in [15] are denoted by a prefix "B-" in the corresponding name and the data were obtained by the data archive that COCO provides.

Observations

Aggregated results over all 24 functions of the suite show the effectiveness of SLSQP. In dimension 5, it has the highest success rate for a budget range $[18D, 800D]$ while in dimension 20, it dominates all solvers up to $\sim 1400D$ function evaluations, after which it is outperformed by L-BFGS-B.

It is interesting to see the performance difference of SLSQP and B-SLSQP in unimodal functions such as the separable Ellipsoid function: in $5D$, the runtimes are almost equal for the easiest targets and then SLSQP is faster by an increasing factor, until termination criteria start to become effective. Performance differences between the early and recent implementation of SciPy, which are not due to the different parameter setting or restart policy of [15], are also observed for BFGS and Nelder-Mead, showing an improvement of the library implementation.

Comparing BFGS and L-BFGS-B, the latter can show better performance for some functions in *all* dimensions, as it is the case for the Sphere, Linear Slope, original and rotated Rosenbrock and Bent Cigar functions. Overall, the picture is more diverse: BFGS has same or higher success rate in the budget ranges $[25D, 125D]$ and $[50D, 400D]$ for dimensions 5 and 20 respectively, while the runtimes always differ less than by a factor of 4.

For Nelder and Mead's method, adaptation of parameters is crucial. Without this option, the algorithm is deteriorated as the dimension increases. In dimension 20, the smallest target values for the Sphere function are not reached while in the aggregated ECDF the method is dominated by all other solvers and for all budgets.

It is interesting that COBYLA, that is based simply on linear interpolation, often achieves better performance for the fraction of easiest targets than all other solvers e.g. for the Sharp Ridge and Sum of Different Powers funtions in $20D$, even outperforming the virtual best solver of BBOB 2009 for small budgets. More remarkable is the performance on the multimodal Gallagher and Katsuura functions in $20D$, where it is one of the most effective methods.

Finally, Differential Evolution shows the best performance among the other (local) solvers for the group of multimodal functions with adequate global structure, where also the Basin Hopping policy is of advantage. Even though the effectiveness of DE weakens with increasing dimensionality, it maintains the highest success rate in this function group.

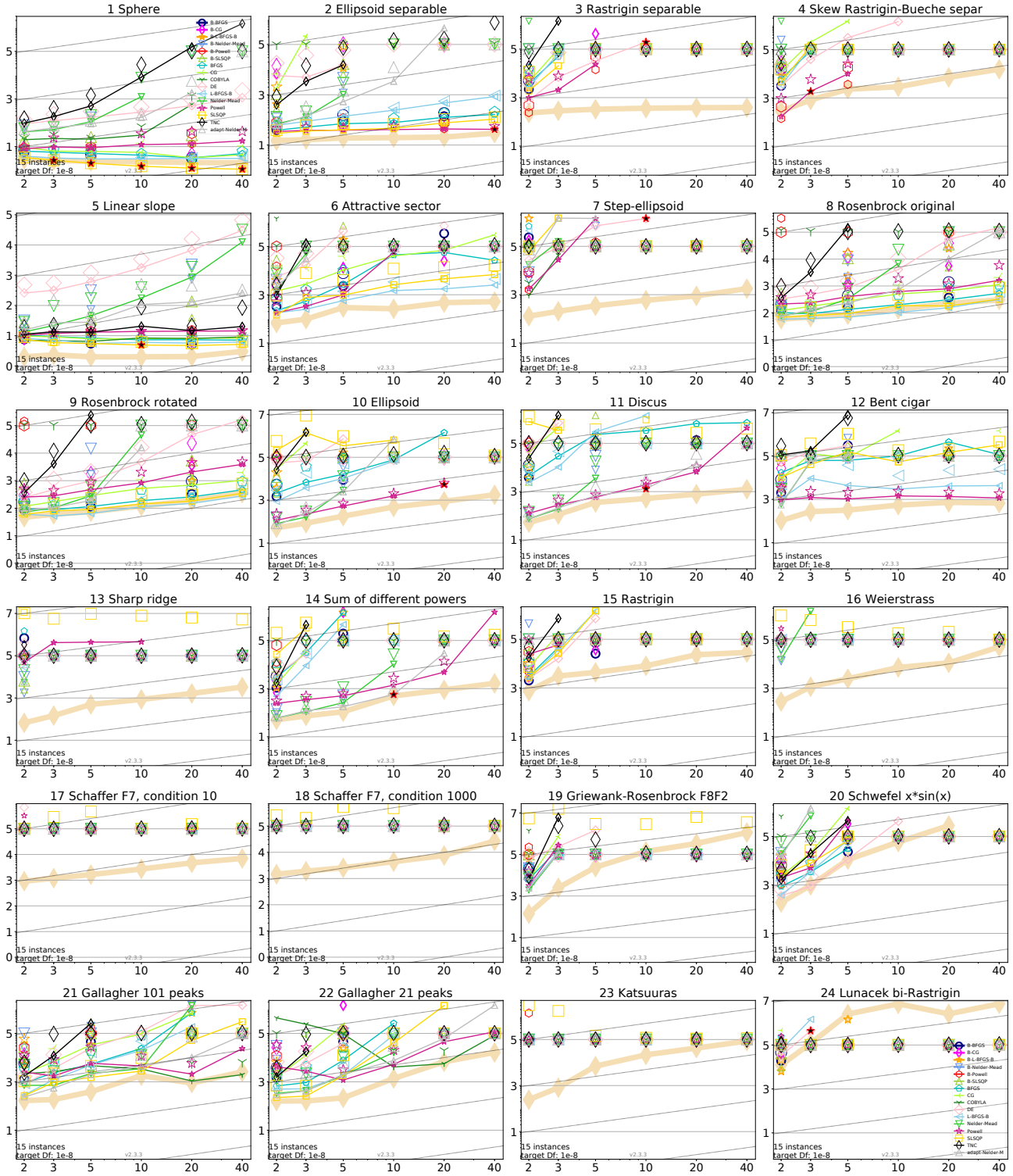

Figure 4.16: Average running time (aRT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: LMCMA, ◇: VkD-CMA, ⋆: LMCMA14, ▽: LMCMA17, ⬡: R10ES, △: R2ES, ⬠: V2D, ⊦: VD, ⊤: VkD, ◇: m2DLBFGS, ◁: sepCMA, ▽: Nelder-Mead, ⋆: Powell, □: SLSQP, ◇: TNC, △: adapt-Nelder-Mead

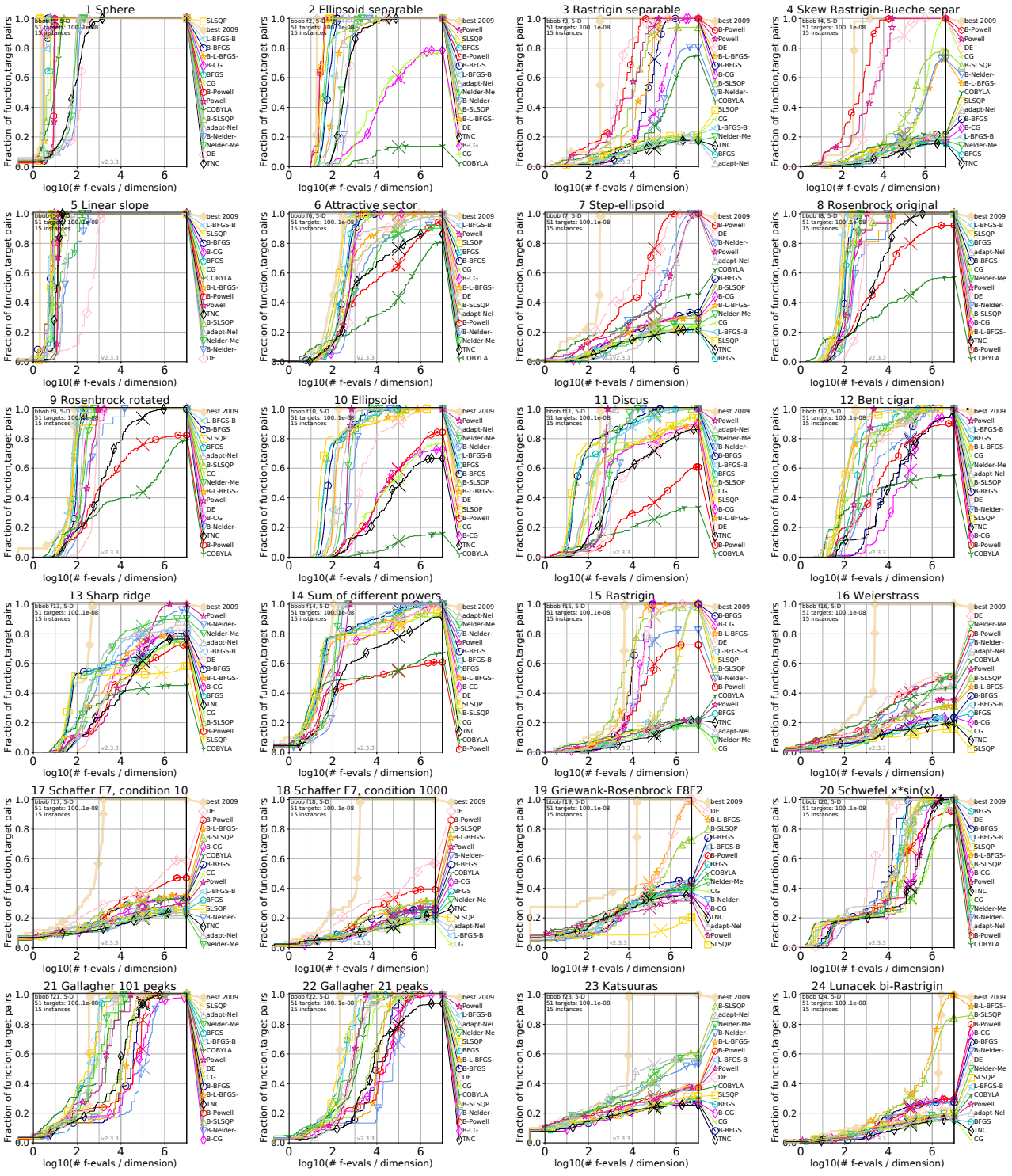

Figure 4.17: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 5.

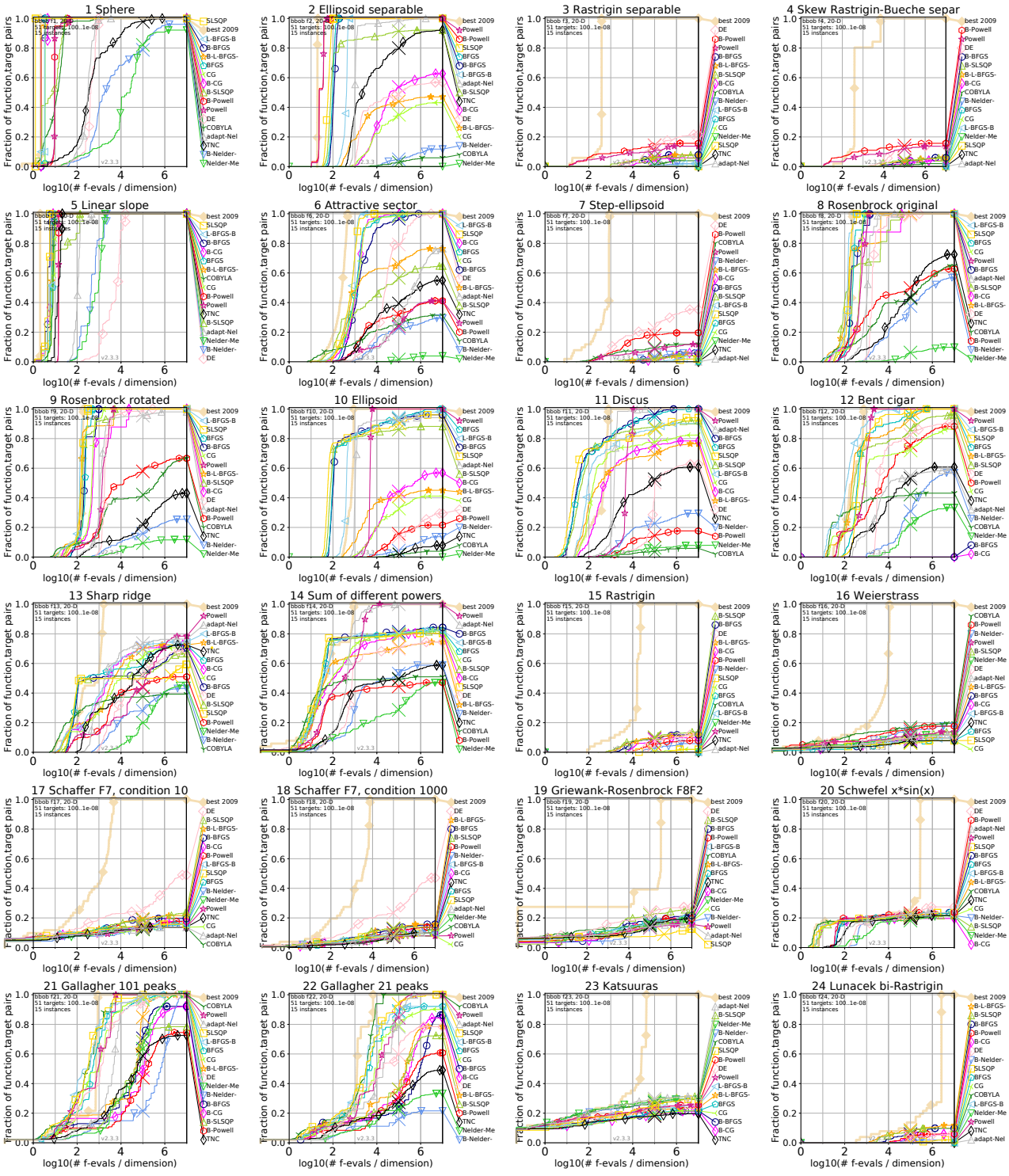

Figure 4.18: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 20.

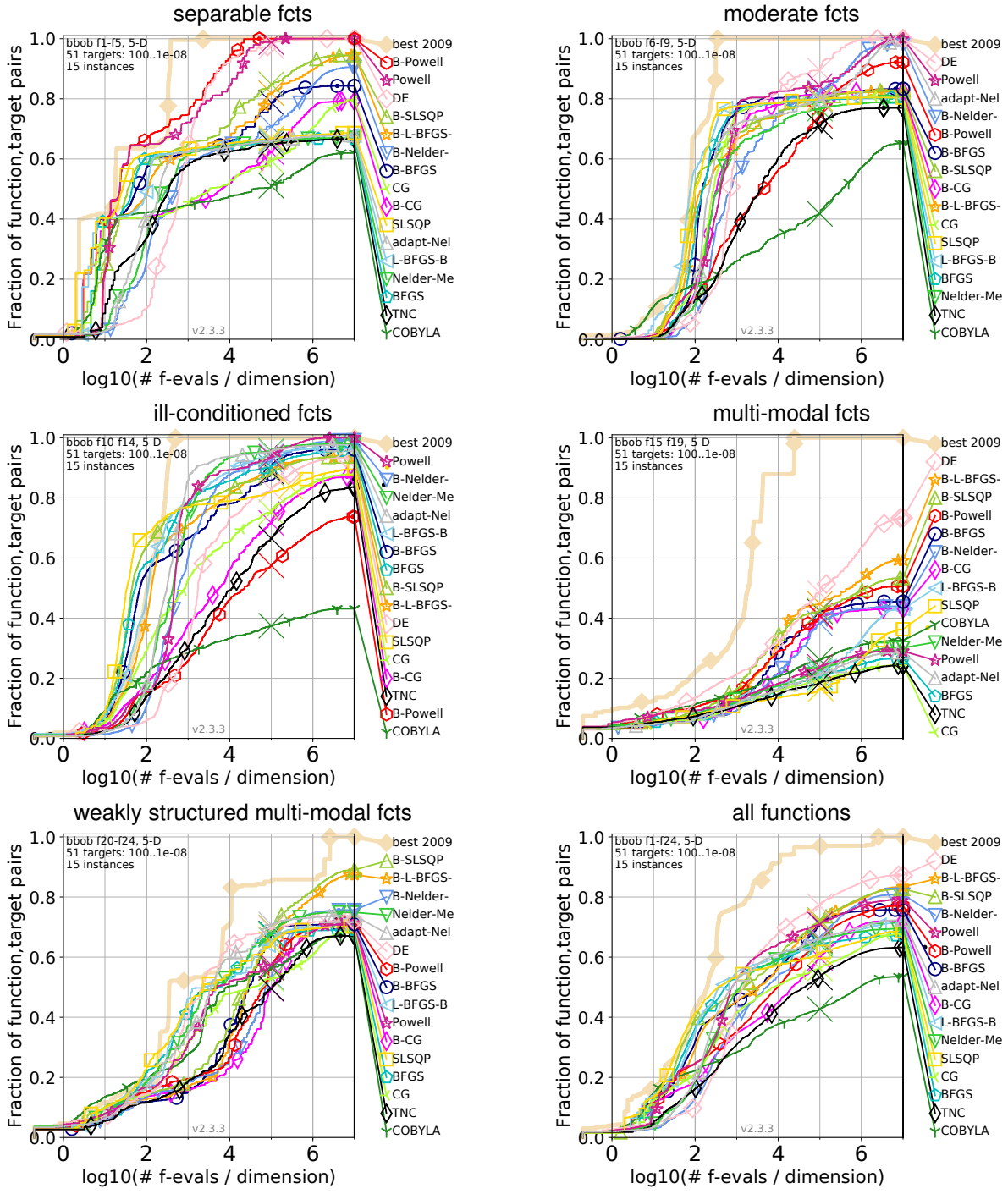

Figure 4.19: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D.

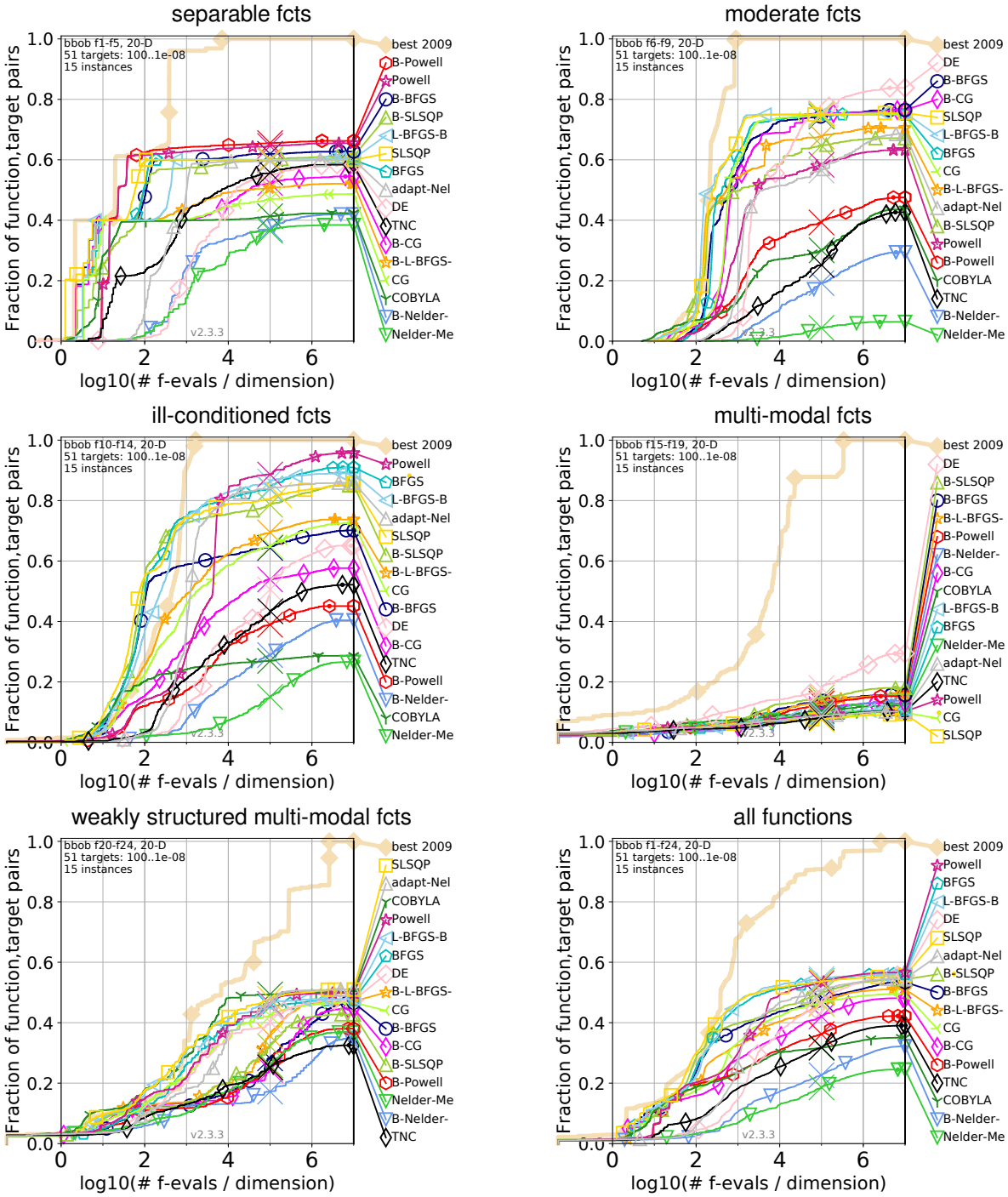

Figure 4.20: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 20-D.

4.3 Discussion

In the first part of this chapter, we described a new benchmarking test suite[17] for black-box optimization up to dimension 640 and based on the existing `bbob` test suite of the COCO platform. In contrast to the `bbob` suite, the new `bbob-largescale` suite has linear computational complexity in the dimension which is achieved by replacing orthogonal matrices with permuted orthogonal block-diagonal matrices, previously proposed in [4]. While the new functions are fully backwards comparable with the functions from the `bbob` test suite, additional adjustments were made (i) to have uniform target values that are comparable in difficulty over a wide range of dimensions, (ii) to have a constant proportion of distinct axes that remain consistent with the `bbob` test suite for the Discus, Bent Cigar and Sharp Ridge functions, and (iii) to not make the Rosenbrock functions significantly easier in higher dimensions due to diminishing distances between the optimum and the search space origin when the dimension increases.

This new suite is a natural extension of the well-established `bbob` suite. By building on the COCO framework with a standardized and established performance assessment procedure, any future benchmarking results can be seamlessly compared with results previously obtained by other researchers. We showcased in the provided user guide how automated benchmarking experiments on the `bbob-largescale` test suite can be performed and gave examples where the graphical output reveals deficiencies of current large-scale optimization algorithms.

Furthermore, in the second part, we showed the obtained results of two studies benchmarking several local solvers both in the small and in the large scale setting, using the `bbob` and the `bbob-largescale` suite respectively. We discussed the importance of several parameter settings of the tested algorithms and compared their performance. All data sets included in this chapter are available online in order to easily compare them with solvers which are to be evaluated with COCO in the future.

[17] The source code is available at `https://github.com/numbbo/coco/blob/master/code-experiments/src/suite_largescale.c` as part of the COCO platform.

Chapter 5

Radar related applications

This chapter is dedicated to two radar-related applications, the problems of phase code optimization and this of phased-array pattern design.

5.1 Phase code optimization

Multiple-input multiple-output (MIMO) radars transmit different signals from each transmit element, which are to be separated in the receiving end. Coherent MIMO radars, in particular, have a geometric configuration with closely spaced elements. The problem of phase code optimization is a waveform design problem, such that waveforms with *good* auto-correlation and cross-correlation properties are found, properties which in turn allow the separation of the received pulses. The former serves in distinguishing a signal from a time-delayed version of itself and the latter for the distinction of different signals, in the case where the pulses transmitted from each element of the coherent MIMO radar are not identical.

5.1.1 Problem formulation

Let $\mathbf{a}$ be a complex valued sequence of length N with values of unit-modulus, i.e. in its polar form $\mathbf{a} = (a_i)_{i=0,\dots,N-1} = (e^{j\phi_i})_{i=0,\dots,N-1}$. Its autocorrelation, denoted here as $\mathbf{c_a}$ is defined as

$$\mathbf{c_a}(k) = \sum_{i\in\mathbb{Z}} \mathbf{a}(i+k)\overline{\mathbf{a}(i)} \tag{5.1}$$

where the sequence $\mathbf{a}$ is zero-padded outside its domain, i.e. we consider $\mathbf{a}(i) = 0$ if $i \notin \{0,\dots,N-1\}$, thus the domain of the autocorrelation sequence $\mathbf{c_a}$ is the set $\{-N+1,\dots,N-1\}$. Similarly, for two sequences $\mathbf{a},\mathbf{b}$, their

cross-correlation, denoted as $\mathbf{c_{a,b}}$, is defined as

$$\mathbf{c_{a,b}}(k) = \sum_{i \in \mathbb{Z}} \mathbf{a}(i+k)\overline{\mathbf{b}(i)}. \tag{5.2}$$

By definition, an autocorrelation sequence $\mathbf{c_a}$ of a unit-amplitude pulse $\mathbf{a}$ has its peak at 0 with a corresponding amplitude equal to $\mathbf{c_a}(0) = N$ for a unit-amplitude sequence $\mathbf{a}$ of length N. The objective is to find phases that define sequences with low autocorrelation sidelobe levels and/or low cross-correlation amplitude levels, i.e. to minimize

$$\mathsf{sll}(\mathbf{c_a}) := \max_{i \neq 0} |\mathbf{c_a}(i)| \tag{5.3}$$

and

$$\mathsf{peak}(\mathbf{c_{a,b}}) := \max_{i} |\mathbf{c_{a,b}}(i)|, \tag{5.4}$$

or in (normalized) dB values

$$\mathsf{sll}^{\mathsf{dB}}(\mathbf{c_a}) := \max_{i \neq 0} 20 \log_{10} \left(\frac{|\mathbf{c_a}(i)|}{N} \right) \tag{5.5}$$

and

$$\mathsf{peak}^{\mathsf{dB}}(\mathbf{c_{a,b}}) := \max_{i} 20 \log_{10} \left(\frac{|\mathbf{c_{a,b}}(i)|}{N} \right). \tag{5.6}$$

Tan et al. [88] address this problem by considering the (weighted) energy of the autocorrelation sidelobes and of the cross-correlation, formulating a smooth objective function and performing a gradient descent method to optimize it. In our setting, we consider the non-smooth problem of minimizing the maximum sidelobe level of the autocorrelation and/or the peak level of the cross-correlation (which additionally does not require any tuning of the weighting function), using black-box CMA-ES based methods. Even though a gradient-based approach for optimizing the weighted energy is feasible and potentially achieves convergence faster than a black-box method, the main difficulty arises due to the multimodality of (both the smooth and the non-smooth) objective function. Preliminary experimentation in certain test cases verified that, in comparison to other local solvers (e.g. [51, 57]) with a random initialization of phases uniformly in $[0, 2\pi]$, a restart policy of a stochastic search method with an increasing size of the sampled population typically offers better quality solutions. This observation is also consistent with several benchmarking results on multimodal functions, where the advantage of increased population sizes was revealed, see also Subsection 2.3.1.

5.1.2 Experimental setting

The following sections illustrate the obtained results when considering single and multiple signals of different lengths. For each optimization problem, a restart strategy [12] with increasing population size (IPOP CMA-ES) has been applied, which is advantageous for multimodal objective functions [39]. For the solution of each problem, the default

(active) CMA-ES solver[1] was used for dimensions smaller than 100, otherwise the VkD-CMA-ES. Restarts were applied until a maximum budget of function evaluations was exceeded. [2]

5.1.3 Results

Single pulse The simplest test case is the autocorrelation optimization of a single signal $\mathbf{a}$ of length N, which consists of a problem also of dimension N with the sequence phases as search variables. The problem reads

$$\underset{\phi_i \in [0,2\pi], i \in \{0,\dots,N-1\}}{\text{minimize}} \ \text{sll}^{\text{dB}}(\mathbf{c_a}). \tag{5.7}$$

Figure 5.1 illustrates the autocorrelation of a sequence with a random choice of phases (uniform in $[0, 2\pi]^N$) in comparison to the optimized autocorrelation, for various lengths N.

Multiple sequences pulse For multiple signals $\mathbf{a}_k,\ k \in \{1,\dots,K\}$ of length N, let us denote ϕ the vector composed of the phases of all K sequences. Then the $N \times K$ - dimensional problem reads

$$\underset{\phi \in [0,2\pi]^{NK}}{\text{minimize}} \max\{\text{sll}^{\text{dB}}(\mathbf{c}_{\mathbf{a}_i}), \text{peak}^{\text{dB}}(\mathbf{c}_{\mathbf{a}_i,\mathbf{a}_j}) : i \neq j, i, j \in \{1,\dots,K\}\}. \tag{5.8}$$

Figure 5.2 illustrates the optimized correlations for the 2-signal cases, where we consider different pulse lengths as in the previous test case. Figure 5.3 shows the obtained results for varying pulse numbers and pulse lengths, in search space dimensions 256 and 512.

Both in the single pulse and in the multiple pulse cases, the IPOP strategy consistently provided better quality solutions in high dimensions when the population size was increased in comparison to the default value, see e.g. Figure 5.4.

5.2 Phased Array pattern design

Phased Array Radars require the design of transmission patterns to achieve illumination or to avoid interference in specific regions. In order to maximize efficiency and sensitivity, the transmit element amplitudes are usually saturated and the beam shape is formed through phase-only control. The pattern design can be viewed as a problem of minimizing a distance to a predefined pattern shape, with the transmit element phases as search variables.

[1] Version 2.7.0 of `pycma`

[2] Five restarts were performed for each problem, with a population size increasing by a factor of 2.

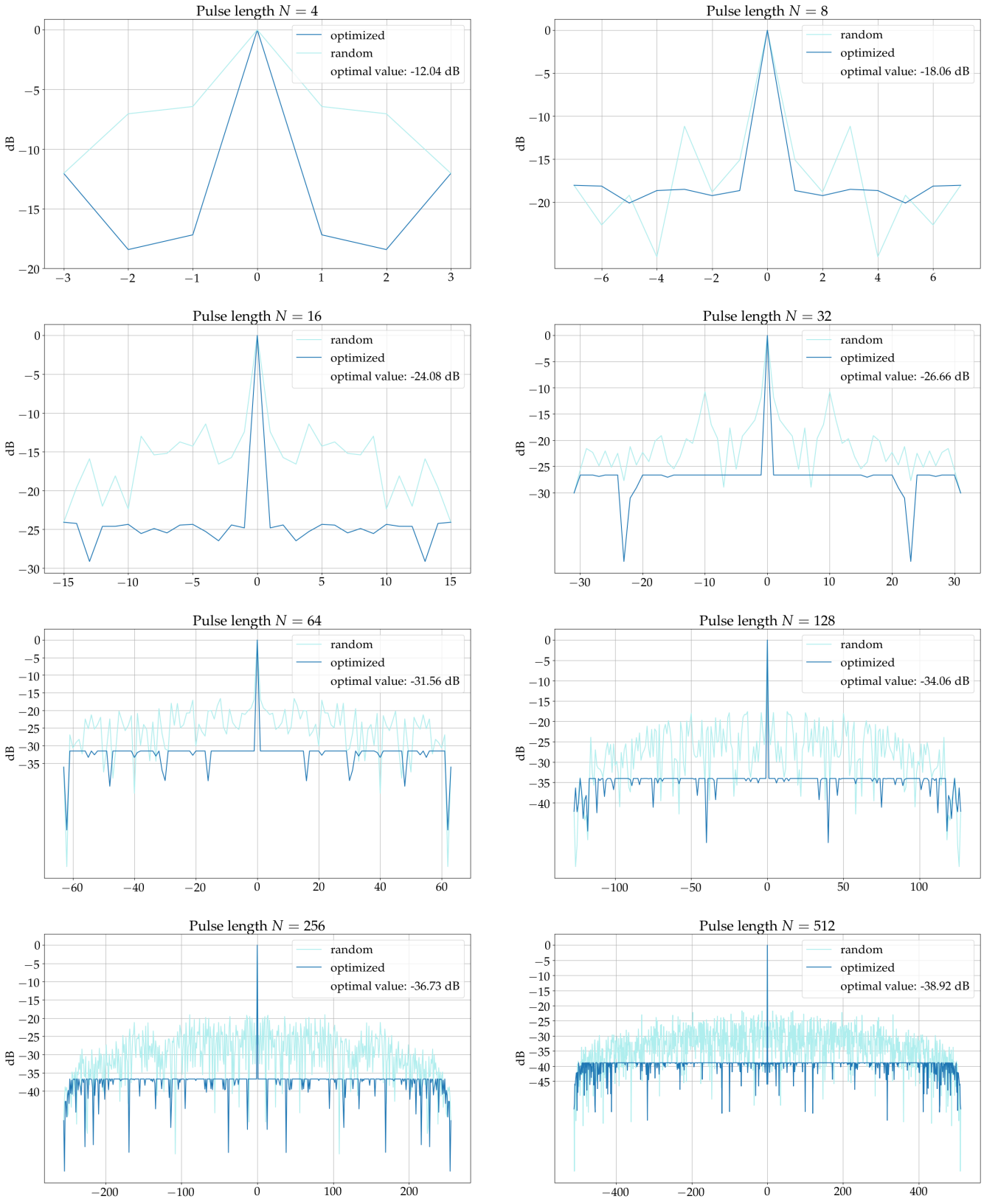

Figure 5.1: Comparison of the optimized autocorrelation of pulses with increasing lengths, with the autocorrelation of pulses with a random choice of phases uniformly in $[0, 2\pi]^N$.

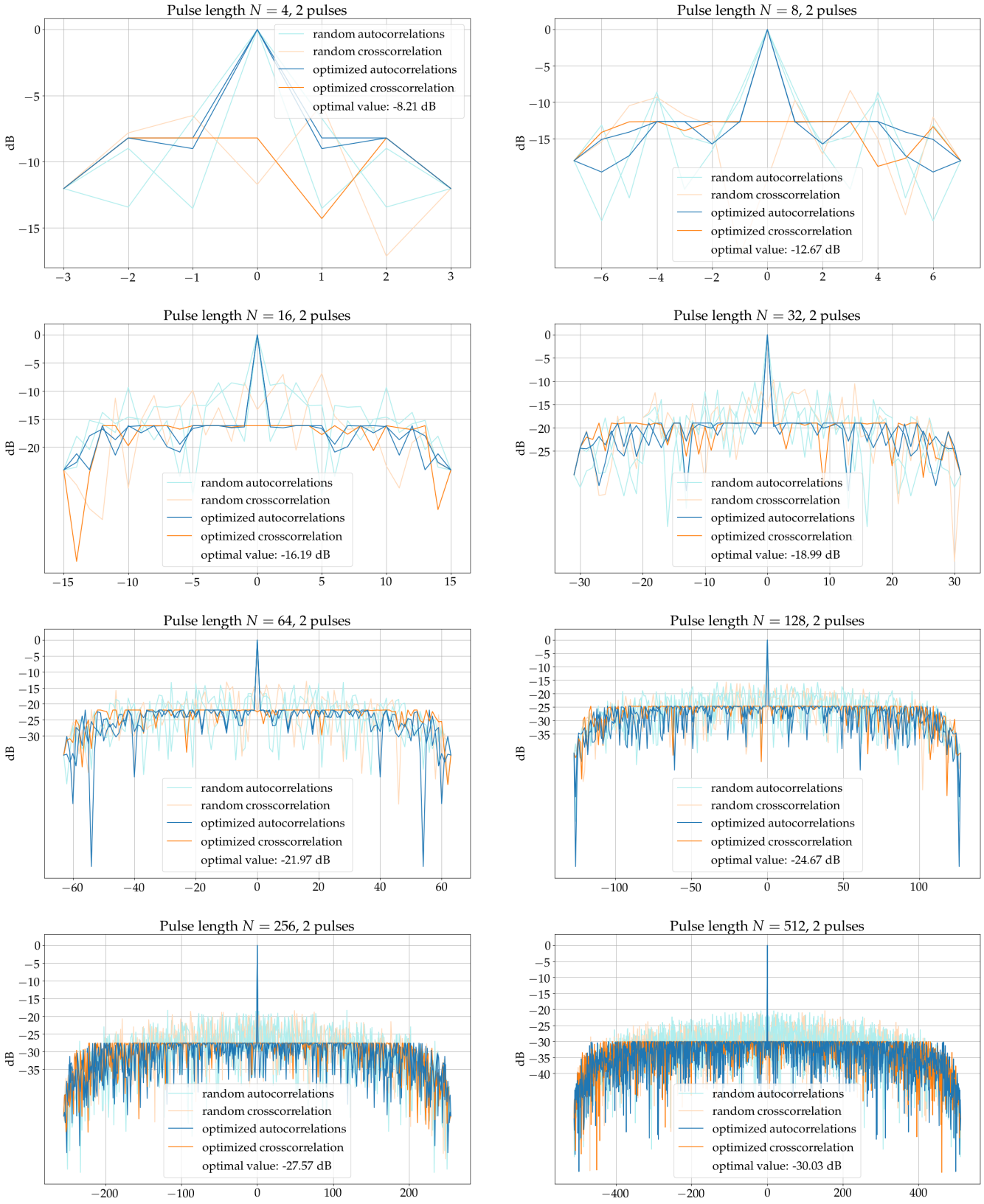

Figure 5.2: Comparison of the optimized autocorrelations and cross-correlation of 2 pulses with increasing pulse lengths, with the autocorrelations and cross-correlation resulting from a random choice of phases uniformly in $[0, 2\pi]^{2N}$.

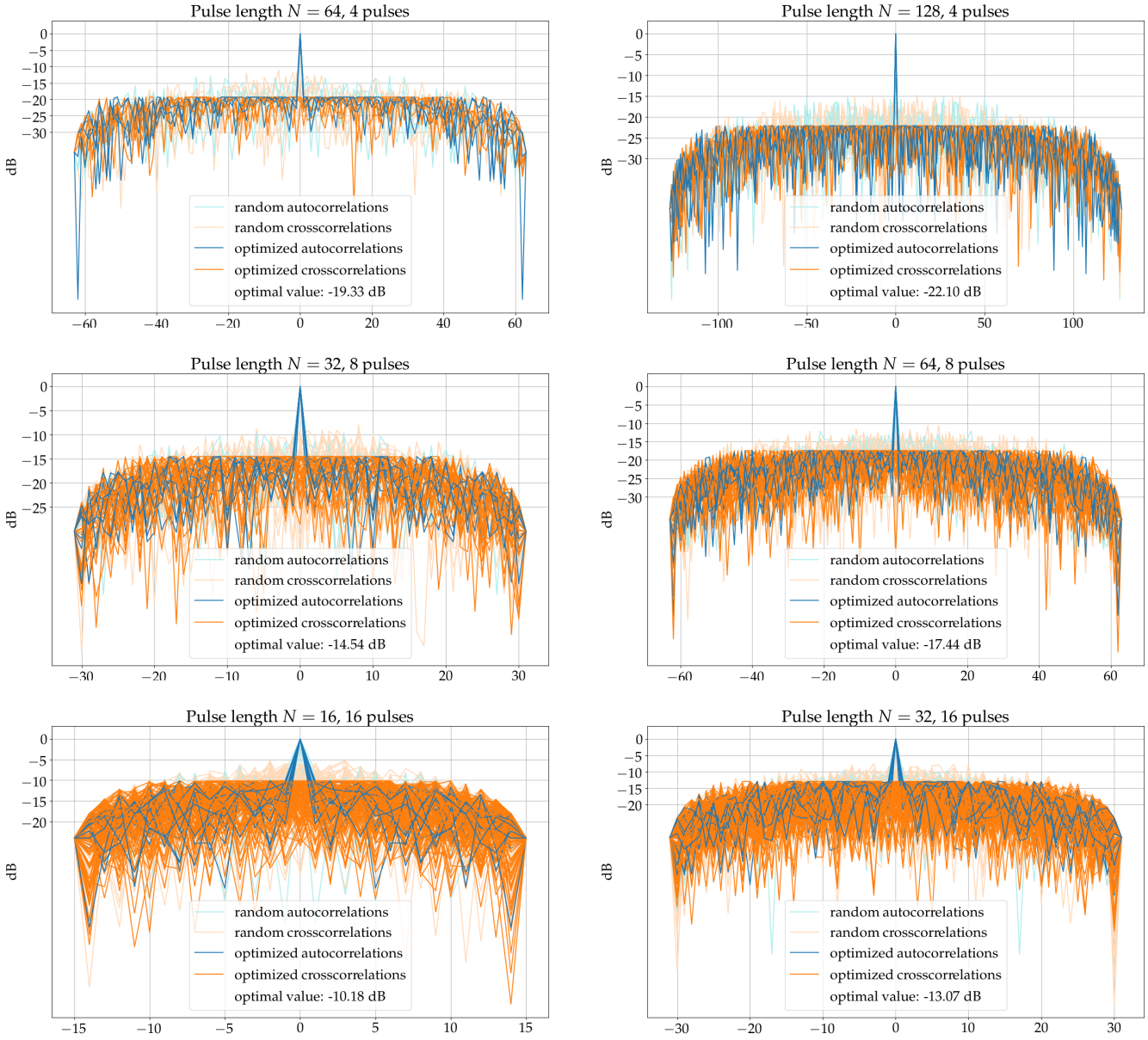

Figure 5.3: Optimized autocorrelations and cross-correlations for 4 (top), 8 (middle) and 16 (bottom) pulses. The search space dimension is equal to 256 (left) and 512 (right), with the corresponding pulse lengths.

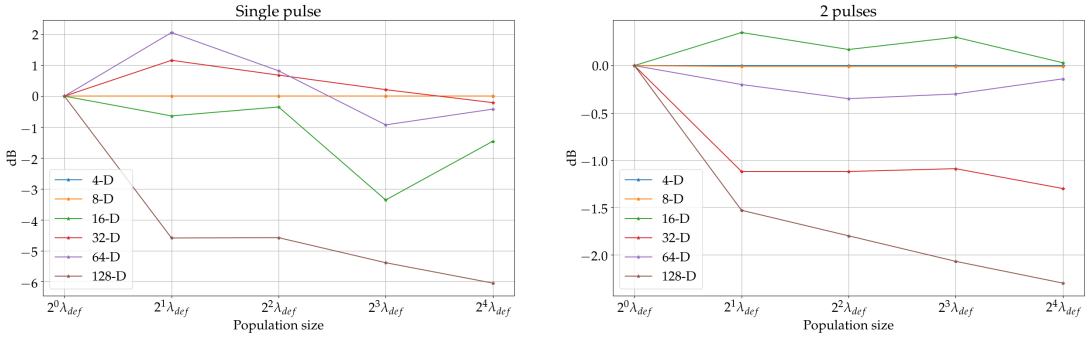

Figure 5.4: Effect of increasing the population size for the single pulse (left) and 2-pulse cases (right). Shown are the differences $f_\lambda^* - f_{\text{def}}^*$ versus the population size, where f_λ^* is the optimal value found for population size λ and f_{def}^* the optimal value for the default population size.

5.2.1 Problem formulation

We consider the cases of planar phased-array antennas with elements located on a grid described by a matrix Λ, i.e. the sensor locations are: $\mathbf{x_n} = \lambda\Lambda\mathbf{n}, \mathbf{n} \in \mathcal{N} \subset \mathbb{Z}^2, \mathbf{x_n} \in \mathcal{X} \subset \mathbb{R}^2$, λ being a positive constant representing the transmission wavelength. Figure 5.5 (top) shows typical phased array grid geometries with the corresponding grid matrices Λ. Each sensor's contribution to the total pattern is determined by its location $\mathbf{x_n}$ and its excitation $a(\mathbf{x_n}) = \alpha_\mathbf{n} e^{j\phi_\mathbf{n}}$, composed by the sensor amplitude $\alpha_\mathbf{n}$ and phase $\phi_\mathbf{n}$. The sensor amplitudes and phases will constitute the search variables in the optimization process. After superposition of each contribution (assuming that the sensors have identical characteristics), the total (far-field) beam shape is described by the Array Factor, a two dimensional function $A : \mathbb{R}^2 \mapsto \mathbb{C}$, defined over the spatial frequency space, as [70]:

$$A(\mathbf{u}) = \sum_{\mathbf{x_n} \in \mathcal{X}} a(\mathbf{x_n}) e^{j\frac{2\pi}{\lambda}\mathbf{u}^T\mathbf{x_n}}. \tag{5.9}$$

The spatial frequency coordinates $\mathbf{u} = \binom{u}{v}$ over the so-called visible space $||\mathbf{u}||_2 < 1$ are related to the azimuth-elevation coordinates as:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \cos el \sin az \\ \sin el \end{pmatrix}, \begin{pmatrix} az \\ el \end{pmatrix} = \begin{pmatrix} \arctan \frac{u}{\sqrt{1-u^2-v^2}} \\ \arcsin v \end{pmatrix}. \tag{5.10}$$

The array factor is periodic with a period Π defined by the matrix Λ^{-T} as $\Pi = \{\Lambda^{-T}\mathbf{v},\ \mathbf{v} \in [0,1]^2\}$. Figure 5.5 (bottom) shows periods of the array factor corresponding to different grid geometries. Additionally, Figure 5.6 (bottom left) depicts a case of an Array Factor function over a domain covering its period (which in this test case is $[0, 2.5]^2$). The goal is to achieve a desired shape of the modulus of the array factor $|A|$ described by a function $D : \mathbb{R}^2 \to \mathbb{R}$. Due to the periodicity of A, we restrict the definition of D in one period Π. Usually, D is defined to represent some desired characteristics in a subset $\Omega \subset \Pi$, for example a flat mainlobe or a zone of zero modulus of A, and in the remaining region $\Pi\backslash\Omega$ the requirement is to sufficiently suppress the sidelobes, therefore, in the following, we refer to Ω as the mainlobe region and to $\Pi\backslash\Omega$ as the sidelobe region. These two goals are competitive and for this, we follow a constraint approach described in the following subsections.

Let E_Ω denote the integrated distance between $|A|$ and D over Ω and $E_{\Pi\backslash\Omega}$ the sidelobe energy of A, i.e.

$$E_\Omega = \left(\int_\Omega (|A(\mathbf{u})| - D(\mathbf{u}))^2 d\mathbf{u} \right)^{1/2} \tag{5.11}$$

and

$$E_{\Pi\backslash\Omega} = \int_{\Pi\backslash\Omega} |A(\mathbf{u})|^2 d\mathbf{u}. \tag{5.12}$$

The quantities E_Ω and $E_{\Pi\backslash\Omega}$ are employed for the mainlobe shaping and the sidelobe suppression respectively. As mentioned above, the search variables are the excitation amplitudes and phases of the sensors $\alpha_\mathbf{n}$ and $\phi_\mathbf{n}$.

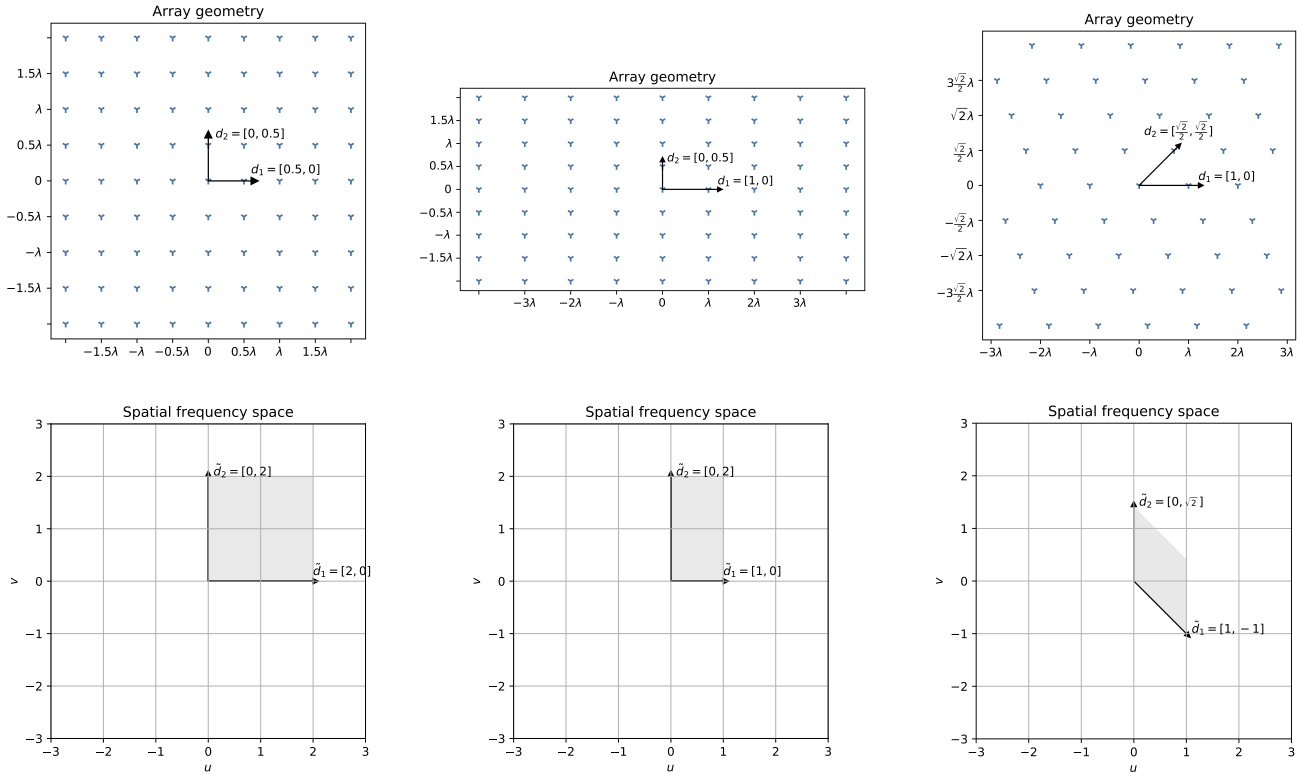

Figure 5.5: Top: Phased arrays with different grid geometries. Bottom: The corresponding periods of the Array Factor are highlighted at the spatial frequency space. In the left column the grid is rectangular, with a grid basis matrix $\Lambda = \begin{pmatrix} d_1^T & d_2^T \end{pmatrix} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$ and the period of the Array Factor is defined by the dual $\Lambda^{-T} = \begin{pmatrix} \tilde{d}_1^T & \tilde{d}_2^T \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$. The grey highlighted domain indicates a period of the Array Factor. Middle and right columns: the grid matrices are $\Lambda = \begin{pmatrix} d_1^T & d_2^T \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0.5 \end{pmatrix}$ with $\Lambda^{-T} = \begin{pmatrix} \tilde{d}_1^T & \tilde{d}_2^T \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ and $\Lambda = \begin{pmatrix} d_1^T & d_2^T \end{pmatrix} = \begin{pmatrix} 1 & \frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} \end{pmatrix}$ with $\Lambda^{-T} = \begin{pmatrix} \tilde{d}_1^T & \tilde{d}_2^T \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & \sqrt{2} \end{pmatrix}$ respectively.

Without any additional constraint, the problem would be trivial: one may simply consider the L_2 projection of D at the space spanned by $\{\Pi \ni \mathbf{u} \mapsto e^{j\frac{2\pi}{\lambda}\mathbf{u}^T\mathbf{x_n}} : \mathbf{x_n} \in \mathcal{X}\}$ (D has to be defined over the whole period Π in this case), i.e. consider the excitations

$$a^*_{\mathbf{n}} = \left| \int_{\Pi} D(\mathbf{u}) \overline{e^{j\frac{2\pi}{\lambda}\mathbf{u}^T\mathbf{x_n}}} d\mathbf{u} \right| \tag{5.13}$$

$$= \left| \int_{\Pi} D(\mathbf{u}) e^{-j2\pi\mathbf{u}^T\Lambda\mathbf{n}} d\mathbf{u} \right| \tag{5.14}$$

and

$$\phi^*_{\mathbf{n}} = \text{Arg} \int_{\Pi} D(\mathbf{u}) \overline{e^{j\frac{2\pi}{\lambda}\mathbf{u}^T\mathbf{x_n}}} d\mathbf{u} \tag{5.15}$$

$$= \text{Arg} \int_{\Pi} D(\mathbf{u}) e^{-j2\pi\mathbf{u}^T\Lambda\mathbf{n}} d\mathbf{u}, \tag{5.16}$$

with the corresponding array factor

$$A^*(\mathbf{u}) = \sum_{\mathbf{x_n}\in\mathcal{X}} a^*_{\mathbf{n}} e^{j\phi^*_{\mathbf{n}}} e^{j\frac{2\pi}{\lambda}\mathbf{u}^T\mathbf{x_n}} \tag{5.17}$$

as the integrated distance minimizer. However, in the case of a phase-only design, the sensors are driven in saturation and we have the additional constraint that the sensor amplitudes are equal. This constraint makes the problem significantly more difficult, due to multimodality of the objective function, as we describe in Subsection 5.2.3.

5.2.2 Integral approximation via FFT

Before moving to the optimization formulation, we remark that all involved integrals are approximated as finite sums, with the discretization of the period Π defined over the points $\mathbf{u_k} = \Lambda^{-T}\mathbf{R}^{-1}\mathbf{k}$, where $\mathbf{R} = \left(\begin{smallmatrix} r_1 & 0 \\ 0 & r_2 \end{smallmatrix}\right)$ and $\mathbf{k} \in \{0, \ldots, r_1 - 1\} \times \{0, \ldots, r_2 - 1\}$. Computing the integrals of equations (5.11) and (5.12) in each iteration as finite sums can be costly, in particular with high frequencies $r_1,\ r_2$. It is possible, though, to use an inverse Fast Fourier Transform routine for the computation of samples of A (D is fixed and only computed once in practice). In [83], a method that achieves this is explained in detail, and it involves zero-padding and data rearrangement of the excitations (in particular one has to mirror the sensor locations $\mathbf{x}_n$ such that they fall to the positive $x-$ and $y-$ half spaces), in order to use a standard 2D-IFFT routine. We follow this approach in all our computations.

5.2.3 Encoding - Initialization - Zeros of the array factor

The equal amplitude constraint is encoded by introducing a search variable α that controls the (w.l.o.g. positive) amplitude, taken as α^2. Note that Parseval's relation

$$\int_{\Pi} |A(\mathbf{u})|^2 d\mathbf{u} = \frac{1}{\det \Lambda} \sum_{\mathbf{n} \in \mathcal{N}} |\alpha_{\mathbf{n}}|^2 \tag{5.18}$$

relates the total array factor energy with the excitation amplitudes, therefore it is useful to introduce α as a search variable for arbitrary definitions of the desired D pattern shape. It is useful here to mention that in [83], the author considers a slightly different formulation compared to our approach: assuming that the amplitude level is fixed to a certain value, say w.l.o.g. equal to 1, the function D in the mainlobe region is scaled such that its total energy is slightly smaller than the energy resulting from Parseval's relation with unit amplitudes. In this approach, the integrated distance of $|A|$ and D over the mainlobe region is considered as the objective function, with the amplitude level fixed and only the phases as search variables. Since the mainlobe energy of D is slightly smaller than the total energy over the whole period Π, and assuming that the Array Factor $|A|$ approximates D in the mainlobe region, the remaining energy is allocated to the sidelobes and therefore if this proportion of remaining energy is sufficiently small, the sidelobes are suppressed. However, the decay factor that determines the sidelobe energy is experimentally selected with trial and error. Introducing the amplitude level α^2 as a search variable allow us to avoid this process.

The objective function of equation (5.11) as a function of the level α and of the phases $\phi_{\mathbf{n}}$ is, for non trivial cases of D, multimodal, and the attractive regions of "bad" local optima are characterised by appearances of zeros of the array factor A over the mainlobe region Ω, with the phase spectrum of A being irregular in neighbouring orbits that encircle these null points [83, 30] (phase transitions from $-\pi$ to π appear in such cases). Therefore, it is important that at the starting point of the optimization run, the mainlobe modulus of A is not close to zero (assuming of course that the desired pattern D is not 0), since if this is the case, typically the search does not escape from the attracting region.

In general, to achieve such an initial pattern depends of course on the geometry of the antenna array. Using the fact, though, that A has the expression of an inverse DFT and that the Fourier transform of $\mathbb{R}^2 \ni (x, y) \mapsto \exp(i\pi(x^2 + y^2))$ is $\mathbb{R}^2 \ni (f_x, f_y) \mapsto i \exp(-i\pi(f_x^2 + f_y^2))$, along with the scaling property of the transform, we may choose the initial phases as $c\|\mathbf{x_n}\|_2^2 / \max_{\mathbf{x_n} \in \mathcal{X}} \|\mathbf{x_n}\|_2^2$ to achieve a relatively smooth initial phase spectrum of A (in this case, a good choice for c would depend on the inter-element spacing of the sensors).

It is also important when using stochastic search methods, to start with a small step size, and to carefully handle the constraint of suppressing the expression of (5.12): for example, if a penalty method or a Lagrangian method is used, optimizing the aggregated objective can lead to such attractive regions even if the starting point is well chosen.

5.2.4 Gradual Sidelobe Energy Suppression

In order to satisfy all the above, we solve a sequence of optimization problems P_κ

$$\begin{aligned} \underset{\alpha,\phi_{\mathbf{n}}}{\text{minimize}} & \left(\int_{\Omega} \left(\left| \sum_{\mathbf{x_n}\in\mathcal{X}} \alpha^2 e^{j\phi_{\mathbf{n}}} e^{j\frac{2\pi}{\lambda}\mathbf{u}^T\mathbf{x_n}} \right| - D(\mathbf{u}) \right)^2 d\mathbf{u} \right)^{1/2} \\ \text{s.t.} & \int_{\Pi\setminus\Omega} \left(\left| \sum_{\mathbf{x_n}\in\mathcal{X}} \alpha^2 e^{j\phi_{\mathbf{n}}} e^{j\frac{2\pi}{\lambda}\mathbf{u}^T\mathbf{x_n}} \right| \right)^2 d\mathbf{u} \leq L^\kappa. \end{aligned} \tag{5.19}$$

For the first run, the problem is unconstrained ($L^0 = \infty$) and it is solved with the initialization of the previous section. This gives us an estimation of a maximal sidelobe energy level E_{max}, and for the next runs we consider $L_\kappa = \epsilon^\kappa E_{\text{max}}$ with $\epsilon \in (0, 1)$, while each problem $\mathsf{P}_{\kappa+1}$ is initialized at the optimal solution of the previous P_κ.

The minimal decay factor $\epsilon^{\kappa_{\text{max}}}$ is experimentally selected in order to have a sufficient balance between the beam shape in Ω and the sidelobe suppression. All experiments related to the shown results used the VkD-CMA solver [6] and the constraints were handled by an Augmented Lagrangian constraints handler [10], implemented within the latest versions of the Python CMA-ES module.

5.2.5 Test Cases

In this final section, we include representative test cases and the corresponding obtained results for different levels of sidelobe energy suppression. Figure 5.6 illustrates the obtained results when the desired pattern is a flat top beam over the domain $||\mathbf{u}|| \leq 0.3$ and 0 elsewhere, for a circular phased array geometry with sensor locations $\mathbf{x}_n = 0.4\lambda \binom{n}{m}, n, m \in \mathbb{Z}$ and $||\mathbf{x}_n/\lambda|| \leq 5$. Figure 5.7 corresponds to a flat top desired mainlobe over an azimuthal domain $|az| \leq \frac{\pi}{16}$, in the case of a rectangular array with sensors located in $\mathbf{x_n} \in \{0.4\lambda(n, m)^T, n, m = -16, \ldots, 16\}$. Lastly, figure 5.8 illustrates details of the obtained pattern when the antenna array has the same (rectangular) geometry as the previous test case and when the desired mainlobe shape is decreasing as $1-\frac{|az|}{\pi/6}$ over the azimuthal domain $|az| \leq \frac{\pi}{6}$ and over the upper hemisphere $v \geq 0$.

5.3 Discussion

We described in this chapter the problems of phase-code waveform design and of phase-only pattern design. Both of them suffer from multimodality, thus we proposed ways to overcome this difficulty. The advantage of CMA-ES in multimodal optimization via an increased population size was crucial for the phase-code problem, in order to obtain better quality solutions than other local solvers with a random initialization. We illustrated the effect of increased population size in increasing dimensions, proving the adequacy of CMA-ES for this Radar problem. Concerning the phase-only pattern design, the crucial part for overcoming its restrictions and difficulties was its proper mod-

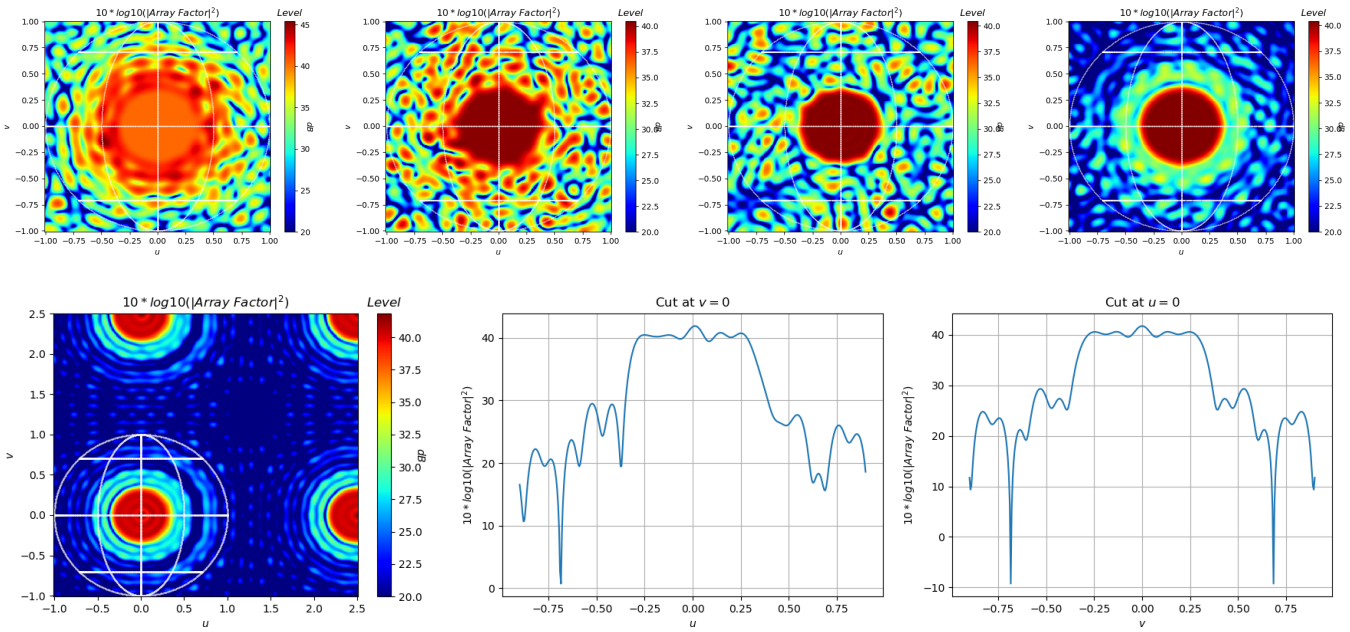

Figure 5.6: Top: Resulting patterns for a circular flat top beam and for decay factors $\epsilon = \infty, 1/2, 1/4, 1/8$ from left to right. Bottom left: Pattern for $\epsilon = 1/16$ over a region including a whole period of A. Bottom right: Details in azimuth/elevation slices.

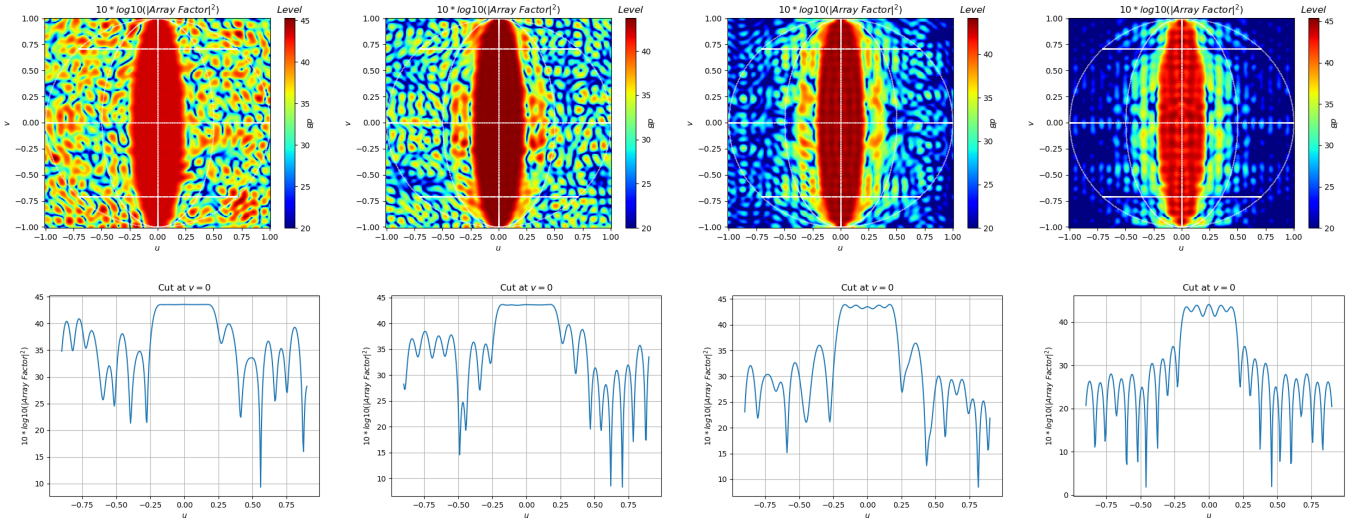

Figure 5.7: Top: Resulting patterns for a flat top beam over the azimuthal domain $|az| \leq \frac{\pi}{16}$ and for decay factors $\epsilon = 1/4, 1/8, 1/16, 1/32$ from left to right. Bottom: Details in azimuth slices. By further suppressing the sidelobes, the mainlobe ripple increases.

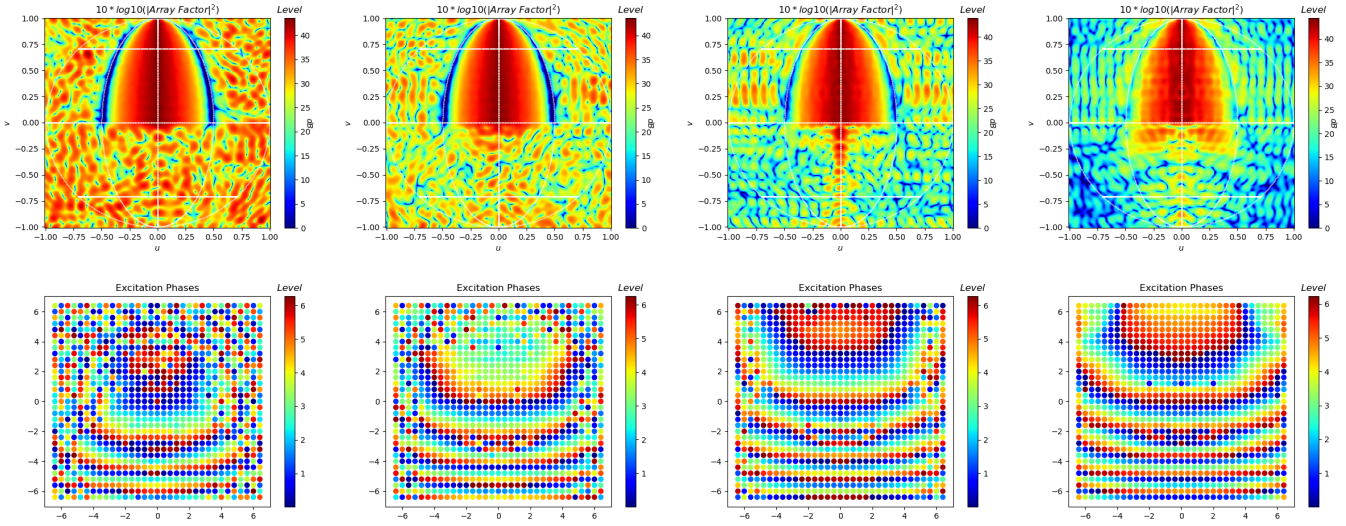

Figure 5.8: Example of a non flat beam shape over the azimuthal domain $|az| \leq \frac{\pi}{16}$ and the upper hemisphere. Top: Resulting patterns decay factors $\epsilon = 1/4, 1/8, 1/16, 1/32$ from left to right. Bottom: Optimized phase excitations modulo 2π of the rectangular phased array.

elling as a sequence of constrained optimization problems. This formulation, combined with a careful initialization, avoids heuristic methods, used e.g. in [83], for the sidelobe suppression and in turn establishes a more principled methodology for designing arbitrary patterns and with various antenna array geometries.

Chapter 6

Discussion

In this thesis we investigated various aspects of stochastic gradient-free optimization related to algorithmic design, benchmarking and real-world applications, with emphasis on methods based on the stochastic search CMA-ES algorithm and for large scale optimization.

We attempted to propose novel ideas of extending CMA-ES for high-dimensional problems with sparsity properties, namely for problems belonging to the class of partial separable functions. In particular, three approaches were introduced and the corresponding limitations were discussed. The technique of hard-thresholding, was discussed due to its simplicity, though under the parameter setting that we presented, the gain in the scaling behaviour was rather poor. Most promising are the approaches of Graphical Lasso regularization as well as the single-link update technique of learning a sparse precision, though further steps are also required. Concerning the former, we illustrated the effect of a uniform penalization (which was originally employed by Graphical Lasso) showing that it cannot be directly integrated in CMA-ES. We proposed an alternative, non-uniform penalization which in turn improved the method's scaling with dimension, depending on the true sparsity properties of test functions. Since in the black box optimization setting, such properties are not known a priori, a better (possibly adaptive) choice of the penalization weights remains an open question for future research. This was also the reason that we did not yet perform the benchmarking of CMA-ES with Graphical Lasso using the COCO platform: the regularization step is computationally expensive and we considered that it is useful to appropriately set the adaptive penalization mechanism before continuing to the large scale benchmarking comparison. For the latter, the link detuning effect is required to be addressed in order to obtain a satisfactory method for arbitrary black-box (dense or sparse ill-conditioned) problems.

The process of comprehending and identifying the advantages and flaws of previously proposed algorithms was necessary for the inspiration of our novel approaches and required their thorough experimental evaluation. Therefore, a significant part of our work was dedicated to various aspects of benchmarking. We discussed the methodology of solvers' benchmarking, focusing on the large scale domain, and finalised the development of the

previously initiated `bbob-largescale` suite of the COCO benchmarking platfrom, describing in detail the platform's experimental and post-processsing parts. Comparative studies of various solvers were additionally performed. We compared among others the most promising variants of CMA-ES in the large scale domain with the prominent L-BFGS algorithm, concluding on the overall scaling behaviour of these methods. Among the CMA-ES variants, none of the previous methods attempted to exploit partial separability, hence our decision to focus on this goal. However, our benchmarking studies were rather extensive and not restricted only to CMA-ES based methods, providing insights also for the real world applications that we tried to address.

We focused on two real world applications related to Radars, those of the phase code optimization problem for coherent MIMO Radars and of the pattern design problem for phased-array Radars. The common difficulty of multi-modality in both applications, along with the common constraint of saturated excitations were addressed in different ways in each case. For the former, as various benchmarking results indicated, the IPOP restart policy of CMA-ES consistently provided solutions of better quality in the test cases that we explored, in particular for increasing dimensions, when the population size was increased. This advantage of CMA-ES over other local solvers proves the importance of its use for addressing this problem. For the latter, a proper modelling of the problem is the crucial part for obtaining satisfactory results. We investigated a sequential pattern refining technique formulated as a constrained optimization problem, in order to avoid heuristic methods [83] for the sidelobe suppression. Furthermore, the characterisation of local optima was particularly important, since it allowed to combine our methodology with a careful choice of initialisation. As a result, this methodology establishes a framework for the design of arbitrary patterns and for various geometries of the phased array Radar.

Bibliography

[1] COCO GitHub issue #1733. `https://github.com/numbbo/coco/issues/1733`, 2018.

[2] numbbo/coco: Comparing continuous optimizers. Getting started. `https://github.com/numbbo/coco#getting-started-`, continuously updating.

[3] M. A. Abramson, C. Audet, J. E. Dennis Jr, and S. L. Digabel. Orthomads: A deterministic mads instance with orthogonal directions. *SIAM Journal on Optimization*, 20(2):948–966, 2009.

[4] O. Ait Elhara, A. Auger, and N. Hansen. Permuted orthogonal block-diagonal transformation matrices for large scale optimization benchmarking. In *Genetic and Evolutionary Computation Conference (GECCO 2016)*. ACM-Press, 2016.

[5] Y. Akimoto and N. Hansen. Projection-based restricted covariance matrix adaptation for high dimension. In *Genetic and Evolutionary Computation Conference (GECCO 2016)*, pages 197–204, Denver, United States, July 2016.

[6] Y. Akimoto and N. Hansen. Online model selection for restricted covariance matrix adaptation. In *Parallel Problem Solving from Nature (PPSN 2016)*, pages 3–13. Springer, 2016.

[7] Y. Akimoto and N. Hansen. Diagonal acceleration for covariance matrix adaptation evolution strategies. *Evolutionary computation*, 28(3):405–435, 2020.

[8] Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi. Bidirectional relation between cma evolution strategies and natural evolution strategies. In *International Conference on Parallel Problem Solving from Nature*, pages 154–163. Springer, 2010.

[9] Y. Akimoto, A. Auger, and N. Hansen. Comparison-based natural gradient optimization in high dimension. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 373–380, 2014.

[10] A. Atamna, A. Auger, and N. Hansen. Augmented lagrangian constraint handling for cma-es?case of a single linear constraint. In *International Conference on Parallel Problem Solving from Nature*, pages 181–191. Springer, 2016.

[11] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2006.

[12] A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *2005 IEEE congress on evolutionary computation*, volume 2, pages 1769–1776. IEEE, 2005.

[13] A. Auger, N. Hansen, and M. Schoenauer. Benchmarking of continuous black box optimization algorithms. *Evolutionary Computation*, 20:481, 2012.

[14] O. Banerjee, L. E. Ghaoui, and A. d?Aspremont. Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data. *Journal of Machine learning research*, 9(Mar):485–516, 2008.

[15] P. Baudiš. Cocopf: An algorithm portfolio framework. *arXiv preprint arXiv:1405.3487*, 2014.

[16] P. J. Bickel, E. Levina, et al. Covariance regularization by thresholding. *The Annals of Statistics*, 36(6): 2577–2604, 2008.

[17] D. Bickson. Gaussian belief propagation: Theory and aplication. *arXiv preprint arXiv:0811.2518*, 2008.

[18] A. Blelly, M. Felipe-Gomes, A. Auger, and D. Brockhoff. Stopping criteria, initialization, and implementations of BFGS and their effect on the bbob test suite. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1513–1517. ACM, 2018.

[19] A. S. Bondarenko, D. M. Bortz, and J. J. Moré. Cops: Large-scale nonlinearly constrained optimization problems. Technical report, Argonne National Lab., IL (US), 2000.

[20] I. Bongartz, A. R. Conn, N. Gould, and P. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software (TOMS)*, 21(1):123–160, 1995.

[21] P. A. Bosman, J. Grahl, and D. Thierens. Benchmarking parameter-free amalgam on functions with and without noise. *Evolutionary computation*, 21(3):445–469, 2013.

[22] A. Bouter, T. Alderliesten, C. Witteveen, and P. A. Bosman. Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 705–712, 2017.

[23] A. Bouter, S. C. Maree, T. Alderliesten, and P. A. Bosman. Leveraging conditional linkage models in gray-box optimization with the real-valued gene-pool optimal mixing evolutionary algorithm. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 603–611, 2020.

[24] A. R. Conn, K. Scheinberg, and P. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. *Approximation theory and optimization: tributes to MJD Powell*, pages 83–108, 1997.

[25] A. R. Conn, K. Scheinberg, and P. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical programming*, 79(1-3):397, 1997.

[26] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*. SIAM, 2009.

[27] A. d'Aspremont, O. Banerjee, and L. El Ghaoui. First-order methods for sparse covariance selection. *SIAM Journal on Matrix Analysis and Applications*, 30(1):56–66, 2008.

[28] A. P. Dempster. Covariance selection. *Biometrics*, pages 157–175, 1972.

[29] J. Fan, Y. Liao, and H. Liu. An overview of the estimation of large covariance and precision matrices. *The Econometrics Journal*, 19(1):C1–C32, 2016.

[30] J. Fienup and C. Wackerman. Phase-retrieval stagnation problems and solutions. *JOSA A*, 3(11):1897–1907, 1986.

[31] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[32] F. Gao and L. Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Comp. Opt. and Appl.*, 51:259–277, 2012.

[33] P. Gilmore and C. T. Kelley. An implicit filtering algorithm for optimization of functions with many local minima. *SIAM Journal on Optimization*, 5(2):269–285, 1995.

[34] N. I. Gould, D. Orban, and P. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

[35] H.-M. Gutmann. A radial basis function method for global optimization. *Journal of global optimization*, 19(3): 201–227, 2001.

[36] N. Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[37] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[38] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009. URL https://hal.inria.fr/inria-00362633.

[39] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Genetic and Evolutionary Computation Conference (Companion)*, pages 1689–1696, New York, NY, USA, 2010. ACM. ISBN 9781450300735.

[40] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2012: Experimental setup. Technical report, INRIA, 2012. URL http://coco.gforge.inria.fr/bbob2012-downloads.

[41] N. Hansen, A. Auger, D. Brockhoff, D. Tušar, and T. Tušar. COCO: Performance assessment. *ArXiv e-prints*, arXiv:1605.03560, 2016.

[42] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. COCO: The experimental procedure. *ArXiv e-prints*, arXiv:1603.08776, 2016.

[43] N. Hansen, D. Brockhoff, O. Mersmann, T. Tusar, D. Tusar, O. A. ElHara, P. R. Sampaio, A. Atamna, K. Varelas, U. Batu, D. M. Nguyen, F. Matzner, and A. Auger. COmparing Continuous Optimizers: numbbo/COCO on Github. Zenodo, Mar. 2019. URL https://github.com/numbbo/coco.

[44] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, pages 1–31, 2020.

[45] F. Herrera, M. Lozano, D. Molina, and M. With. Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems, 2010.

[46] K. Holmström. An adaptive radial basis algorithm (arbf) for expensive black-box global optimization. *Journal of Global Optimization*, 41(3):447–464, 2008.

[47] G. A. Jastrebski and D. V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *2006 IEEE international conference on evolutionary computation*, pages 2814–2821. IEEE, 2006.

[48] A. Kabán, J. Bootkrajang, and R. J. Durrant. Toward large-scale continuous eda: A random matrix theory perspective. *Evolutionary computation*, 24(2):255–291, 2016.

[49] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[50] J. N. Knight and M. Lunacek. Reducing the space-time complexity of the CMA-ES. In *Genetic and Evolutionary Computation Conference (GECCO 2007)*, pages 658–665. ACM, 2007. ISBN 978-1-59593-697-4.

[51] D. Kraft. A software package for sequential quadratic programming. Tech. rep. dfvlr-fb 88-28, 1988.

[52] O. Krause, D. R. Arbonès, and C. Igel. CMA-ES with optimal covariance update and storage complexity. In *NIPS Proceedings*. 2016.

[53] J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *arXiv preprint arXiv:1904.11585*, 2019.

[54] J. Laska and M. Narayan. skggm 0.2.7: A scikit-learn compatible package for Gaussian and related Graphical Models, July 2017. URL `https://doi.org/10.5281/zenodo.830033`.

[55] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin. Benchmark functions for the CEC'2013 special session and competition on large scale global optimization. Technical report, Evolutionary Computation and Machine Learning Group, RMIT University, Australia, 2013.

[56] Z. Li and Q. Zhang. A simple yet efficient evolution strategy for large scale black-box optimization. *IEEE Transactions on Evolutionary Computation*, 2017.

[57] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(3):503–528, Dec. 1989. ISSN 0025-5610.

[58] I. Loshchilov. A computationally efficient limited memory CMA-ES for large scale optimization. In *Genetic and Evolutionary Computation Conference (GECCO 2014)*, pages 397–404, 2014. ISBN 978-1-4503-2662-9.

[59] I. Loshchilov. LM-CMA: an alternative to L-BFGS for large scale black-box optimization. *Evolutionary Computation*, 25:143–171, 2017.

[60] I. Loshchilov, T. Glasmachers, and H. Beyer. Limited-memory matrix adaptation for large scale black-box optimization. *CoRR*, abs/1705.06693, 2017.

[61] M. Lozano, D. Molina, and F. Herrera. Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing*, 15:2085–2087, 2011. doi: 10.1007/s00500-010-0639-2.

[62] V. Martin, C. Furtlehner, Y. Han, and J.-M. Lasgouttes. Gmrf estimation under topological and spectral constraints. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 370–385. Springer, 2014.

[63] R. Mazumder and T. Hastie. Exact covariance thresholding into connected components for large-scale graphical lasso. *The Journal of Machine Learning Research*, 13(1):781–794, 2012.

[64] J. Moré and S. Wild. Benchmarking Derivative-Free Optimization Algorithms. *SIAM J. Optimization*, 20 (1):172–191, 2009. Preprint available as Mathematics and Computer Science Division, Argonne National Laboratory, Preprint ANL/MCS-P1471-1207, May 2008.

[65] S. G. Nash. Newton-type minimization via the lanczos method. *SIAM Journal on Numerical Analysis*, 21(4): 770–788, 1984.

[66] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[67] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[68] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[69] Y. Ollivier, L. Arnold, A. Auger, and N. Hansen. Information-geometric optimization algorithms: A unifying picture via invariance principles. *The Journal of Machine Learning Research*, 18(1):564–628, 2017.

[70] S. J. Orfanidis. Electromagnetic waves and antennas, 2002. URL `http://eceweb1.rutgers.edu/~orfanidi/ewa/`.

[71] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[72] P. Pošík and V. Klemš. JADE, an adaptive differential evolution algorithm, benchmarked on the BBOB noiseless testbed. In *Genetic and Evolutionary Computation Conference (Companion)*, pages 197–204, New York, NY, USA, 2012. ACM. ISBN 9781450311786.

[73] M. J. Powell. Recent research at cambridge on radial basis functions. In *New Developments in Approximation Theory*, pages 215–232. Springer, 1999.

[74] M. J. Powell. Uobyqa: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

[75] M. J. Powell. The newuoa software for unconstrained optimization without derivatives. In *Large-scale nonlinear optimization*, pages 255–297. Springer, 2006.

[76] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 01 1964. ISSN 0010-4620. doi: $10.1093/\mathrm{comjnl}/7.2.155$. URL `https://doi.org/10.1093/comjnl/7.2.155`.

[77] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*, pages 51–67. Springer, 1994. ISBN 978-94-015-8330-5. doi: $\mathrm{https://doi.org/10.1007/978\text{-}94\text{-}015\text{-}8330\text{-}5\_4}$.

[78] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[79] K. Price. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pages 153–157, Piscataway, NJ, USA, 1997. IEEE. doi: 10.1109/ICEC.1997.592287.

[80] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[81] R. Ros and N. Hansen. A simple modification in CMA-ES achieving linear time and space complexity. In *Parallel Problem Solving from Nature (PPSN 2008)*, pages 296–305. Springer, 2008.

[82] R. Salomon. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278, 1996.

[83] D. P. Scholnik. A parameterized pattern-error objective for large-scale phase-only array pattern design. *IEEE Transactions on Antennas and Propagation*, 64(1):89–98, 2015.

[84] R. R. Steffen Finck, Nikolaus Hansen and A. Auger. Real-parameter black-box optimization benchmarking 2010: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE, Fachhochschule Vorarlberg, Austria, 2009. URL `https://coco.gforge.inria.fr/downloads/download16.00/bbobdocfunctions.pdf`. errata in 2019.

[85] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[86] Y. Sun, F. J. Gomez, T. Schaul, and J. Schmidhuber. A linear time natural evolution strategy for non-separable functions. *CoRR*, abs/1106.1998, 2011.

[87] T. Suttorp, N. Hansen, and C. Igel. Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, 2009.

[88] U. Tan, C. Adnet, O. Rabaste, F. Arlery, J.-P. Ovarlez, and J.-P. Guyvarch. Phase code optimization for coherent mimo radar via a gradient descent. In *2016 IEEE Radar Conference (RadarConf)*, pages 1–6. IEEE, 2016.

[89] R. Tanabe and A. Fukunaga. Tuning differential evolution for cheap, medium, and expensive computational budgets. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2018–2025. IEEE, 2015.

[90] K. Tang, X. Yao, P. Suganthan, C. MacNish, Y. Chen, C. Chen, and Z. Yang. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Technical report, University of Science and Technology of China, 2007.

[91] K. Tang, X. Li, P. Suganthan, Z. Yang, and T. Weise. Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical report, University of Science and Technology of China, 2009.

[92] D. Thierens. The linkage tree genetic algorithm. In *International Conference on Parallel Problem Solving from Nature*, pages 264–273. Springer, 2010.

[93] D. Thierens and P. A. Bosman. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 617–624, 2011.

[94] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on optimization*, 7(1):1–25, 1997.

[95] K. Varelas. Benchmarking large scale variants of cma-es and l-bfgs-b on the bbob-largescale testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1937–1945, 2019.

[96] K. Varelas and M.-A. Dahito. Benchmarking multivariate solvers of scipy on the noiseless testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1946–1954, 2019.

[97] K. Varelas, A. Auger, D. Brockhoff, N. Hansen, O. A. ElHara, Y. Semet, R. Kassab, and F. Barbaresco. A comparative study of large-scale variants of cma-es. In *International Conference on Parallel Problem Solving from Nature*, pages 3–15. Springer, 2018.

[98] K. Varelas, A. Auger, and N. Hansen. Sparse inverse covariance learning for cma-es with graphical lasso. In *International Conference on Parallel Problem Solving from Nature*, pages 707–718. Springer, 2020.

[99] K. Varelas, O. A. El Hara, D. Brockhoff, N. Hansen, D. M. Nguyen, T. Tušar, and A. Auger. Benchmarking large-scale continuous optimizers: the bbob-largescale testbed, a coco software guide and beyond. *Applied Soft Computing*, 97:106737, 2020.

[100] D. J. Wales and J. P. Doye. Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

[101] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(27):949–980, 2014. URL `http://jmlr.org/papers/v15/wierstra14a.html`.

[102] X. Yu and M. Gen. *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.

ECOLE
DOCTORALE
DE MATHEMATIQUES
HADAMARD

Titre : Optimisation sans dérivées stochastique via CMA-ES et Techniques Sparses - Applications Radars

Mots clés : optimisation, boîte-noire, grande échelle, radar

Résumé : Dans cette thèse, nous étudions certains aspects des méthodes aléatoires adaptatives pour l'optimisation continue sans gradient. Les algorithmes que nous étudions sont basés sur sur l'adaptation de la matrice de variance-covariance d'une stratégie évolutionnaire (CMA-ES) et se concentrent sur des problèmes d'optimisation en grande dimension.
Nous commençons par une description de l'algorithme CMA-ES et ses connexions avec l'optimisation géométrique de l'information (IGO), suivie d'une étude comparative des variantes de CMA-ES pour l'optimisation en grande dimension. Nous proposons en outre de nouvelles méthodes qui intègrent des outils d'estimation parcimonieuse de la matrice de variance-covariance afin d'obtenir des algorithmes basés sur CMA-ES plus efficaces pour des problèmes partiellement séparables en grande dimension.
De plus, nous décrivons la méthodologie pour évaluer la performance des algorithmes adoptée par la plateforme Comparing Continuous Optimizers (COCO), et finalisons la suite de problèmes-tests `bbob-largescale`, une nouvelle suite d'analyse comparative d'algorithmes d'optimisation pour des problèmes en grande dimension avec un faible coût de calcul.
Enfin, nous présentons la formulation d'un problème d'optimisation, l'algorithme proposé et les résultats obtenus pour deux applications radar, le problème de recherche de codes de phase pour le filtrage adapté et le problème de synthèse des faisceaux dans une antenne réseau à commande de phase (Phased-Array antenna).

Title : Randomized Derivative Free Optimization via CMA-ES and Sparse Techniques - Applications to Radars

Keywords : optimization, black-box, large scale, radar

Abstract : In this thesis, we investigate aspects of adaptive randomized methods for black-box continuous optimization. The algorithms that we study are based on the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm and focus on large scale optimization problems.
We start with a description of CMA-ES and its relation to the Information Geometric Optimization (IGO) framework, succeeded by a comparative study of large scale variants of CMA-ES. We furthermore propose novel methods which integrate tools of high dimensional estimation within CMA-ES, to obtain more efficient algorithms for large scale partially separable problems.
Additionally, we describe the methodology for algorithm performance evaluation adopted by the Comparing Continuous Optimizers (COCO) platform, and finalize the `bbob-largescale` test suite, a novel benchmarking suite with problems of increased dimensions and with a low computational cost.
Finally, we present the formulation, methodology and obtained results for two applications related to Radar problems, the Phase Code optimization problem and the Phased-Array Pattern design problem.

Institut Polytechnique de Paris
91120 Palaiseau, France